Форма № 24

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ
Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікацій
(повне найменування факультету)

Кафедра інформаційної безпеки та наноелектроніки
(повне найменування кафедри)

Пояснювальна записка

до дипломного проєкту (роботи)
магістр
(ступінь вищої освіти)

на тему Дослідження методів аналізу мережевого трафіку та проєктування системи виявлення загроз у реальному часі
(назва теми)

Виконав(ла): студент(ка) 2 курсу,
групи БК-814м
Спеціальності 125 Кібербезпека та захист інформації
(код і найменування спеціальності)
Освітня програма (спеціалізація)
Безпека інформаційних і комунікаційних систем
ІНОЗЕМЦЕВ.А.Є
(ПРІЗВИЩЕ та ініціали)

Керівник КОРОТУН.А.В
(ПРІЗВИЩЕ та ініціали)
Рецензент ЛИТВИЦЬКИЙ.О.П
(ПРІЗВИЩЕ та ініціали)

2025

Форма № 25

МІНІСТЕРСТВО ОСВІТИ І НАУКИ УКРАЇНИ

Національний університет «Запорізька політехніка»

Факультет інформаційної безпеки та електронних комунікації

Кафедра інформаційної безпеки та наноелектроніки

Ступінь вищої освіти: магістр

Спеціальність125 Кібербезпека та захист інформації

Освітня програма (спеціалізація) Безпека інформаційних і комунікаційних систем

ЗАТВЕРДЖУЮ

Завідувач кафедри ІБтаН, к.ф-м.н.

Андрій КОРОТУН

“____” _____2025 року

З А В Д А Н Н Я

НА ДИПЛОМНИЙ ПРОЄКТ (РОБОТУ) СТУДЕНТА

ІНОЗЕМЦЕВА Андрія Євгеновича

(ПРІЗВИЩЕ, ім’я, по батькові)

1. Тема проєкту (роботи): Дослідження методів аналізу мережевого трафіку та проєктування системи виявлення загроз у реальному часі.Research on methods of network traffic analysis and design of a real-time threat detection system

керівник проєкту (роботи) канд. фізико-математичних. наук, доцент КОРОТУН Андрій Віталійович

(науковий ступінь, вчене звання, ПРІЗВИЩЕ, ім’я, по батькові)

затверджені наказом закладу вищої освіти від « 26 » листопада 2025 року № 530

2. Строк подання студентом проєкту (роботи): 20.12.2025

3. Вихідні дані до проєкту (роботи): теоретичні відомості щодо бібліотек;теоретичні відомості про алгоритми, створення політик ,створення алгоритму,тестування

4. ЗМІСТ розрахунково-пояснювальної записки (перелік питань, що їх потрібно розробити): Опрацювати бібліотеки; Обрати алгоритм на якому буде будуватися програма та обґрунтувати свій вибір; Розробка політик безпеки та алгоритму

5. Перелік графічного матеріалу (з точним зазначенням обов’язкових креслень, кількість слайдів, плакатів): Презентація у програмі Microsoft Power ,скріншоти про створення проєкту і тестування.

6. Консультанти розділів проєкту (роботи)

Розділ	ПРІЗВИЩЕ, ініціали та посада консультанта	Підпис, дата	
		завдання видав	Прийняв виконане завдання
1-3	КОРОТУН. А.В., доцент каф. ІБтаН	05.09.2025	12.12.2025
нормконтроль	КОРОЛЬКОВ Р.Ю., доцент каф. ІБтаН		14.12.2025

7. Дата видачі завдання « 05 » вересня 2025 року.

КАЛЕНДАРНИЙ ПЛАН

№ з/п	Назва етапів дипломного проєкту (роботи)	Строк виконання етапів проєкту (роботи)	Примітка
1	Ознайомлення із завданням, підбір літератури	1 тиждень	Виконано
2	Аналіз загальної відомостей щодо інформації про віртуалізацію.	1 тиждень	Виконано
3	Встановлення бібліотек.	2 тиждень	Виконано
4	Написання робочого білду	3-5 тиждень	Виконано
5	Збірка програми.	5-6 тиждень	Виконано
6	Дебагінг.	6 тиждень	Виконано
7	Тестування.	6 тиждень	Виконано
8	Оформлення пояснювальної записки.	8 тиждень	
9	Написання робочого проекту.	9 тиждень	

Студент(ка) _____ Андрій ІНОЗЕМЦЕВ
… (підпис) (Ім’я ПРІЗВИЩЕ)

Керівник проєкту (роботи) _____ Андрій КОРОТУН
(підпис) (Ім’я ПРІЗВИЩЕ)

АНОТАЦІЯ

Пояснювальна записка до магістерської роботи:107 с.,17 рис., 1 дод.,31 джерело.

IDS, IPS, PYTHON, АНАЛІЗ ТРАФІКУ, ВЕБАНАЛІТИКА, ГІБРИДНА СИСТЕМА, КІБЕРБЕЗПЕКА, МАШИННЕ НАВЧАННЯ, МЕРЕЖЕВА БЕЗПЕКА, СПОВІЩЕННЯ.

Об’єктом дослідження є процеси виявлення та запобігання вторгненням у сучасних комп’ютерних мережах, а також архітектурні принципи побудови інтелектуальних систем безпеки реального часу. У роботі аналізуються методи дослідження мережевого трафіку, підходи до створення гібридних IDS/IPS та механізми інтеграції модулів вебаналітики й сповіщень.

Метою проєкту є опанування теоретичних основ та практична реалізація на мові Python гібридної системи, що поєднує сигнатурний та аномальний аналіз. Наукова новизна полягає у розробці комплексної архітектури, яка забезпечує високу ефективність виявлення відомих загроз та гнучкість у ідентифікації невідомих атак. Додатково впроваджено модуль вебаналітики для візуалізації стану безпеки та систему оперативних сповіщень.

Дипломний проєкт складається з чотирьох розділів.

У першому розділі подано теоретичні відомості про IDS та IPS, наведено класифікацію методів аналізу трафіку, розглянуто сучасні мережеві загрози та проблеми традиційних підходів до побудови систем безпеки.

У другому розділі здійснюється проєктування гібридної моделі: описано архітектуру системи, логіку взаємодії модулів, методи обробки трафіку, принципи функціонування сигнатурного та аномального рушіїв, а також особливості інтеграції вебаналітики та системи сповіщень.

У третьому розділі представлено програмну реалізацію створеної моделі на Python, що включає модулі захоплення трафіку, сигнатурний аналіз, аномальну детекцію, вебінтерфейс, REST API та механізми сповіщення, а також наведено результати тестування системи у різних сценаріях.

У четвертому розділі здійснюється тестування проєкту.

ABSTRACT

Explanatory note to the master's thesis: 107 p., 17 fig., 1 appendix, 31 sources.

IDS, IPS, PYTHON, TRAFFIC ANALYSIS, WEB ANALYTICS, HYBRID SYSTEM, CYBERSECURITY, MACHINE LEARNING, NETWORK SECURITY, NOTIFICATION.

The object of the research is the processes of detecting and preventing intrusions in modern computer networks, as well as the architectural principles of building intelligent real-time security systems. The work analyzes methods of studying network traffic, approaches to creating hybrid IDS/IPS, and mechanisms for integrating web analytics and notification modules.

The goal of the project is to master the theoretical foundations and practical implementation in Python of a hybrid system that combines signature and anomaly analysis. The scientific novelty lies in the development of a comprehensive architecture that provides high efficiency in detecting known threats and flexibility in identifying unknown attacks. Additionally, a web analytics module for visualizing the security status and a system of operational notifications have been implemented.

The work is structured into four sections. The first section presents theoretical information about IDS/IPS, a classification of traffic analysis methods, and a critical review of modern threats. The second section is devoted to the design of the model: a description of the architecture, the logic of the interaction of the modules, and the principles of operation of the signature and anomaly engines. The third section describes the software implementation of the system in Python, including traffic capture components, REST API, and web interface. The fourth section contains the results of comprehensive testing of the project in various operating scenarios.

ЗМІСТ

ПЕРЕЛІК СКОРОЧЕНЬ

AD – Active Directory

HIDS – Host-based IDS

IDS –Intrusion Detection System

IP – Internet Protocol address

IPS – Intrusion Prevention System

NIDS – Network-based IDS

NIST SP 800-115 – National Institute of Standards and Technology Special Publication 800-115

OSSTMM –Open Source Security Testing Methodology Manual

OWASP – Open Web Application Security Project

PTES – Penetration Testing Execution Standard

Raw – Raw Packet

RDP – Remote Desktop Protocol

TCP – Transmission Control Protocol

UDP – User Datagram Protocol

YAML – YAML Ain’t Markup Language

ПЗ – Програмне забезпечення

ВСТУП

В XXI столітті під час стрімкої цифровізації бізнес-процесів комп'ютерні мережі стали основною ціллю для взлому та отримання даних з них. Хакери часто застосовують комплексні атаки які можуть зруйнувати мережу яку створювали під конкретні задачі, під час практичних сценаріїв пентесту було виявлено, що хакери користуються Nmap як сканер для збору інформації про мережеву інфраструктуру, Metasploit а також Mimikatz для отримання облікових даних.

Традиційні міжмережеві екрани не забезпечують достатній рівень безпеки та захисту від подібних широких атак. В той же час, комерційні системи IDS і IPS є дорогими, витратними що потребують високої кваліфікації від спеціаліста із кіберзахисту та налаштування їх інтеграції в систему та обслуговування. Це створює для спеціалістів і бізнесу потребу для створення змінної, вигідної та універсального рішення для моніторингу і опору сучасним загрозам у системі.

Одним із перспективних напрямів у вирішенні даної проблеми є створення власної конструкційної IDS/IPS системи з використанням мови програмування Python. Вибір Python є обґрунтованим не лише через наявність бібліотек як Scapy, Flask, а також і через розвинених засоби автоматизації та також можливість використання машинного навчання.

Метою даної роботи є проєктування та програмна реалізація тестування гібридної моделі IDS/IPS системи на Python, здатної в реальному часі аналізувати трафік, виявляти в ньому загрози та аномалії на основі сигнатурних та химерних ознак, здійснювати активне блокування шкідливої активності та формувати візуальний звіт з використанням веб-інтерфейсу.

Для досягнення поставленої мети необхідно виконати наступні завдання:

– провести аналіз існуючих рішень IDS IPS та критеріїв до виявлення атак;

– дослідити бібліотеки та програмні засоби на Python для аналізу трафіку й

реалізації механізмів виявлення загроз;

– було розроблено архітектуру та модульну гібридну структуру;

– реалізувати програмний прототип системи та протестувати його в реальному мережевому середовищі;

– після розробки необхідно оцінити ефективність роботи нашої системи та розробимо рекомендації щодо її подальшого вдосконалення .

1 АНАЛІЗ ПРЕДМЕТНОЇ ОБЛАСТІ ТА МЕТОДІВ ВИЯВЛЕННЯМЕРЕЖЕВИХ ВТОРГНЕНЬ

Перш ніж почати необхідно розібратися ,що нам потрібно та які інструменти ми будемо використовувати.

Основна роботи це не тільки підтвердити факт, але і довести, що дана проблема є реальною . Буде розглянено, як сучасні багатоетапні атаки легко руйнують традиційні захисні засоби на кшталт звичайних фаєрволів. Це і є обґрунтування актуальності проблеми та довести, що старі методи більше не працюють і потрібні нові, розумніші підходи.

IDS (Intrusion Detection System) є системою, яка, по суті, є сигналізацією, що лише сповіщає про вторгнення і чим вона кардинально відрізняється від IPS (Intrusion Prevention System) яка не лише помічає, але й намагається негайно зупинити або заблокувати загрозу.

Математична модель мережевого трафіку

$$T = \{p_1\ p_2\ p_3\ p_4\ p_5\ p_n\} \tag{1.1}$$

де кожен пакет

$$p_i = (src_i, dst_i, proto_i, size_i, time_i) \tag{1.2}$$

Функція аналізу трафіку IDS:

$$F: T \to A \tag{1.3}$$

де

$$A = \{log, alert, drop, block\} \tag{1.4}$$

1.1 Використання бібліотек Python

Вибір мови Python в якості основа для розробки гібридної IDS/IPS обґрунтований не лише її низьким порогом входження, але й наявністю потужної екосистеми спеціалізованих бібліотек. Вони дозволяють реалізувати швидке створення прототипу та поєднати низькорівневий аналіз мережі з високорівневою веб аналітикою в єдиному програмному комплексі.

Стек технологій, обраний для реалізації проєкту, можна логічно розділити на функціональні групи, що відповідають ключовим модулям системи.

Мережевий аналіз та захоплення пакетів здійснюватимуться з використанням бібліотеки для модуля збору Sniffer scapy. Вона була обрана завдяки своїй унікальній здатності не лише пасивно сканувати трафік в реальному часі, але й можливості аналізувати та модифікувати під час роботи. На відміну від простіших бібліотек, scapy надає об'єктно-орієнтований доступ до будь-якого шару пакета IP, TCP, UDP, Raw/payload, що є фундаментальною вимогою для подальшого сигнатурного та аномального аналізу.

Реалізація правил детекції — цей вбудований модуль є ядром сигнатурного рушія, що використовується для компіляції та пошуку шаблонів заданих у правилах, безпосередньо у байтовому вмісті пакетів. Це дозволяє виявляти специфічні сигнатури атак, такі як команди mimikatz або UNION SELECT запити.

Модуль Collections сприяє та його реалізація аномального рушія, використовує високопродуктивні структури даних defaultdict та deque. Зв'язка defaultdict дозволяє миттєво групувати пакети за IP-адресою джерела без додаткових перевірок. А сама deque забезпечує швидке додавання нового часу та видалення застарілого , що дає нам оптимізувати його роботу в реальному часі

Для активного реагування та взаємодія з ОС Subprocess та Platform під час цієї реалізації функції IPS активно використовується модуль subprocess.

Модуль platform використовується для визначення типу ОС , що забезпечує крос-платформовість механізму та блокування при динамічному виборі між iptables у Linux та netsh advfirewall у Windows. OS та Signal в цьому модулі os застосовується для безпечного зчитування конфігураційних даних токенів Telegram-бота зі змінною оточення, що запобігає збереженню секретних даних у коді.A signal дозволяє коректно обробляти системні сигнали зокрема, SIGINT від Ctrl+C для граційного завершення роботи системи та очищення правил брандмауера.

Візуалізація, сповіщення та API цей мікро-фреймворк було обрано для реалізації модуля вебаналітики. Він дозволяє швидко запустити легкий веб сервер, який одночасно виконує дві функції, яка надає REST API для передачі даних про тривоги та заблоковані IP та віддає HTML-сторінку дашборду для візуалізації моніторингу в реальному часі.

Requests використовується для відправки миттєвих сповіщень у Telegram. Вона є стандартом для виконання HTTP-запитів та ідеально підходить для відправки POST-запитів до Telegram Bot API.

JSON — це вбудований модуль, що являє собою стандарт для обміну даними між бекендом та фронтендом JavaScript в дашборді, а також для ведення структурованих логів атак .

Під час конфігурації та оркестрації буде використано YAML ,обраний для файлу правил ids_rules.yaml завдяки його легкісті сприйняття для людини. Це значно спрощує адміністрування системи та додавання нових сигнатур аналітиками без необхідності редагування коду.

Threading — це модуль для забезпечення паралельної роботи компонентів системи.Оскільки scapy.sniff є блокуючою операцією, то вебсервер та фоновий процес очищення тимчасових блокувань `_block_reaper` запускаються в окремих потоках.

Argparse надає стандартизований та професійний інтерфейс командного рядка CLI для запуску скрипту, дозволяючи користувачу гнучко налаштовувати роботу вибір інтерфейсу, активацію режиму IPS чи API.

Datetime та Time забезпечують всю роботу з часовими мітками та фіксацію точного часу атаки для логів, розрахунок ковзного вікна в deque та керування тривалістю тимчасових блокувань IP-адрес.

Raw Sockets — стандартні мережеві додатки які працюють через сокети типу `SOCK_STREAM` (TCP) або `SOCK_DGRAM` (UDP), де операційна система автоматично обробляє заголовки пакетів. Для IDS потрібний доступ до канального рівня Link Layer, що реалізується через сімейство сокетів `AF_PACKET` у Linux або драйвер Npcap у Windows. Це дозволяє отримувати повний Ethernet-фрейм до його обробки ядром ОС.

Критично важливими для проєкту та системи безпеки є можливість на цьому рівні проводити точний аналізMAC-адреси ARP-пакетів, VLAN-тегування (802.1Q), а також виявлявленя атак, спрямовані на маніпуляцію ARP-cashe, ARP-spoofing або MAC-flooding. Raw-socket забезпечує операцію читання фрейма як бітової послідовності.

$$\mathrm{F} = \{\mathrm{b}_1, \mathrm{b}_2 \ldots\ldots \mathrm{b}_\mathrm{n}\} \tag{1.5}$$

де кожен біт входить у множину $\{0,1\}$. Далі ця послідовність декодується у структуре поля Ethernet-заголовка

$$\mathrm{F} = (\mathrm{MAC}_{\mathrm{src}}, \mathrm{MAC}_{\mathrm{dst}}, \mathrm{EtherType}, \mathrm{Payload}) \tag{1.6}$$

Математично операція декодування може бути описана як відображення

$$D: \{0,1\}^n \rightarrow H_{L2} \tag{1.7}$$

H_{L2} — це простір заголовків канального рівня,що надає доступ IDS виконувати операції порівняння, хешування або аномального аналізу без втручання TCP/IP-стека

ОС. На основі цих даних формується модель оцінки загрозливого пакета. І ймовірність цього пакету, є шкідливим, буде оцінена через функцію Байєса:

$$P(A|F) = \frac{P(F|A)P(A)}{P(F)} \quad (1.8)$$

де A — це подія , а F — це спостережуваний фрейм.

Це дозволяє створювати комбіновані гібридні моделі, де аналіз сигнатур та аномалій працюють разом, підсилюючи точність. Після декодування фрейм переходить у модуль статистичного аналізу, який функціонує як система з ковзним часовим вікном. Часовий ряд пакетів від одного джерела можна подати як функцію потоку подій $N(t)$.

Інтенсивність мережевого потоку оцінюється як

$$\lambda(t) = \frac{dN(t)}{dt} \quad (1.9)$$

Для практичних умов, де значення дискретні, використовується така оцінка, яка система фіксує, що λ перевищує порогову величину, визначену або вручну, або через статистичну модель, то поведінка інтерпретується як аномалія. Важливо, що ковзне вікно deque реалізує модель марковського процесу нульового порядку, де кожен новий елемент витісняє найстаріший, забезпечуючи постійну часову актуальність даних. У сигнатурному аналізі застосовується модель кінцевого автомата.

Регулярний вираз, наприклад для SQL-ін’єкцій “UNION SELECT”, компілюється у автомат

$$A = (\ , \Sigma\ , \delta\ , q\ 0\ , F\ ) \tag{1.10}$$

де Σ — це множина байтів, а функція переходів δ визначає рух між станами автомата при збігу символів. Автомат переходить у кінцевий стан лише при повному збігу сигнатури, що математично гарантує детерміновану поведінку й відсутність хибних спрацьовувань при часткових збігах. Для аномального аналізу застосовуються також принципи машинного навчання. Система формує багатовимірний вектор ознак пакета:

$$\vec{x} = (x_1, x_2, x_3, x_n) \tag{1.11}$$

де $\mathrm{x}_1 = \mathrm{size}, \mathrm{x}_2 = \mathrm{ttl}, \mathrm{x}_3 = \mathrm{flags}, \mathrm{x}_4 = \mathrm{inter} - \mathrm{arriva} - \mathrm{time}$

Далі оцінка аномальності може виконуватися через метрику відстані:

$$\mathrm{D}(\vec{\mathrm{x}}, \vec{\mu}\ ) = \sqrt{\sum_{\mathrm{i}=1}^{\mathrm{n}} (\mathrm{x_i} - \mu_\mathrm{i})^2} \tag{1.12}$$

де $\vec{\mu}$ — це вектор середніх значень нормальної поведінки. Якщо $D > D_{crit}$ то пакет вважається підозрілим. У модулі IPS вводяться формальні моделі керування блокуваннями. Нехай множина заблокованих IP визначається як $B(t)$. Тоді операція блокування це додавання IP до множини

$$\mathrm{B(t + e)} = \mathrm{B(t)U\{ip\}} \tag{1.13}$$

Операція автоматичного розблокування через таймер формально

$$B(t) = \{ip \in B : T_{block}(ip) + \Delta T > t\} \quad (1.14)$$

де ΔT — це конфігурований термін блокування. Таким чином, блокувальник працює як автомат із подіями і часом, де вхідні атаки викликають зміни стану системи фільтрації.

Для REST-API Flask використовується модель запитів/відповідей, яку можна описати як відображення

$$R: C \rightarrow S \quad (1.15)$$

де C — це множина HTTP-клієнтів, а S –це дані IDS/IPS.

Формат JSON описується деревом ключ–значення

$$T = \{ki: vi\} \quad (1.16)$$

де значення можуть бути числами, масивами або вкладеними структурами. Це дозволяє формувати дашборд реального часу, де графіки оновлюються з частотою, обмеженою лише продуктивністю браузера. Модуль Threading забезпечує модель паралельного виконання, яку можна описати через систему.

$$T = \{t1, t2, t3\} \quad (1.17)$$

де t 1 — це Sniffer, $t2$ — це вебсервер, а $t3$ — це блок-reaper. Вони взаємодіють через спільну пам’ять, але із запобіганням гонок за рахунок GIL Python або додаткових блокувань.

Паралельна система може бути описана як граф виконання, де кожен потік має власний розклад:

$$Sched = \{(t_i, \tau_j)\} \tag{1.19}$$

де τ_j — це часовий інтервал, коли потік активний

Модулі datetime і time забезпечують точну побудову часових рядів подій IDS. Кожен запис журналу є кортежем

$$L = (t, ip, event, meta) \tag{1.20}$$

що дозволяє будувати хронологічні карти атак і виконувати аналітику типу time-to-compromise.

Нарешті, уся система IDS/IPS може бути описана як композиція функцій

$$System = API\ IPS\ Detection\ Capture \tag{1.21}$$

де кожна функція перетворює дані на новий рівень абстракції. Така формальна модель показує, що система є не просто набором модулів, а цілісною структурою з чіткою математичною логікою обробки мережевого трафіку та прийняття рішень.

1.2 Актуальність задачі моніторингу та захисту корпоративних мереж

Практичні дослідження демонструють, що проведення атаки на типову корпоративну інфраструктуру, побудовану на базі AD та Windows Server, є тривіальною задачею за наявності поширених інструментів. Зловмисники активно використовують готові методології пентесту для імітації атак.

Традиційні міжмережеві екрани не здатні протистояти таким багатоетапним атакам, оскільки вони оперують на рівні портів та IP-адрес, але не аналізують вміст трафіку та запросів. Виникає критична вразливість яка створює розрив між вартістю та складністю комерційних систем потребують вторгнень (IDS) і нагальною потребою бізнесу в гнучкому, адаптивному та ефективному вирішенні проблеми. Готові open-source рішення, так як Snort або Suricata, є гарним вибором, але вимагають високої кваліфікації для налаштування та написання правил і політик.

У сучасних корпоративних середовищах обробляються великі обсяги чутливої інформації такі як:

- фінансові транзакції
- персональні дані клієнтів
- конфіденційні документи,
- бізнес-аналітика,
- службові
- протоколи доменної інфраструктури.

Все це робить мережу підприємства стратегічно важливим активом, який одночасно є і найбільш привабливою ціллю для багатовекторних атак. Еволюція загроз призвела до того, що класичні міжмережеві екрани, орієнтовані виключно на контроль портів, IP-адрес та простих сигнатур, перестали бути достатніми. Хакери застосовують складні методики обходу захисту, використовують розгалужені Kill Chain-ланцюжки, автоматизовані інструменти, техніки експлуатації вразливостей

ядра Windows, lateral movement у доменних середовищах, обхід аутентифікації та приховування слідів.

Зважаючи на це, важливим для забезпечення кібербезпеки є розробка адаптивної системи, яка зможе інтегрувати в собі як можливості високорівневого моніторингу для аналізу мережевого трафіку, також механізми оперативного блокування загроз, які реагують в режимі реального часу. Проєктована система IDS/IPS на Python створена саме виконати цю функцію та забезпечити гнучку платформу, яка здатна одночасно виконувати пасивне виявлення та активне запобігання атакам .

Сучасні атаки мають високий рівнень автоматизації, за використанням фреймворків на кшталт Metasploit, Cobalt Strike, Empire та повним набором засобів розвитку атаки від первичного сканування до захоплення доменного контролера.

Для демонстрації масштабу проблеми розглянемо, наскільки типовими стали такі етапи атаки:

- розвідка та збір інформації, зловмисники використовують Nmap, Masscan чи ZMap, і здійснюють повне профілювання зовнішніх та внутрішніх сегментів мережі. Скануються критичні сервіси, такі як SMB (порт 445), RDP (3389), LDAP (389/636), Kerberos (88), WinRM (5985/5986), RPC Endpoint Mapper (135) тощо. Відсутність адекватної сегментації мережі або неправильна конфігурація фаєрволів створює можливість для швидкого та майже непомітного визначення топології мережі.
- експлуатація вразливості — однією з найбільш поширених атак залишається експлуатація MS17-010 (EternalBlue), що дозволяє здійснити довільне виконання коду на сервері або робочій станції. Використання Metasploit Framework робить атаку доступною навіть для мало досвідчених хакерів.
- закріплення в системі — після успішного проникнення зловмисник отримує високі привілеї NT AUTHORITY\SYSTEM через Meterpreter,що відкриває

можливості встановлення бекдорів, виконання команд, інтеграції проксі-тунелів та запуску власних імплантів.

- отримання облікових даних — до таких інструментів відноситься,як Mimikatz, яка дозволяє знімати хеші NTLM, витягувати квитки Kerberos, обходити LSASS-захист, виконувати атаки Pass-the-Hash / Pass-the-Ticket. Компрометація одного адміністративного акаунта здатна призвести до повного захоплення цілого домену.
- горизонтальне переміщення застосовує WMI, PsExec, SMB-relay, IPv6-spoofing або DCSync-атаки, зловмисник переходить між системами, отримуючи все більший контроль аж до доменного контролера.

З огляду на наведені сценарії видно, що загроза не лише в окремих атаках, а в повній автоматизації їх виконання. Саме тому традиційні підходи до захисту не забезпечують необхідного рівня безпеки.

Системам необхідна здатність:

- налізувати поведінку трафіку,
- застосовувати машинне навчання для профілювання нормальної активності,
- виявляти аномальні послідовності запитів,
- блокувати підозрілі дії автоматично.

Розробка власної гібридної IDS/IPS на Python покликана вирішити ці завдання. Python обрано завдяки його високій продуктивності у швидкій розробці, наявності бібліотек для роботи з мережами (Scapy, dpkt, socket), інструментів для машинного навчання (sklearn, TensorFlow, PyTorch), можливості інтегрувати механізми блокування (iptables/netsh) та простій побудові веб-панелей (Flask, FastAPI).

Таке рішення є надзвичайно актуальним для молодих фахівців (junior-рівня), які тільки починають працювати в сфері кібербезпеки. На відміну від складних систем Snort, Suricata, Bro/Zeek, запропонована система має значно нижчий поріг входу, прозору архітектуру та високу модульність. Це дозволяє легко розширювати її

функціонал, експериментувати з новими моделями ML, доповнювати власними правилами та механізмами реакції.

Таким чином, актуальність роботи зумовлена необхідністю створення сучасного інструменту, який поєднує в собі:

- глибокий аналіз мережевого трафіку
- моделі машинного навчання для виявлення аномалій
- сигнатурний пошук відомих загроз атак
- активне блокування загроз у реальному часі
- інтегрований веб-інтерфейс для адміністрування
- адаптивність до різних сценаріїв корпоративних мереж

1.3 Класифікація та архітектура систем виявлення вторгнень

Система виявлення та запобігання вторгненням (IDS/IPS) є ключовим елементом сучасної інформаційної безпеки та складається з програмних і апаратних засобів, призначених для виявлення несанкціонованої активності в комп'ютерній мережі та реагування на неї. Основна мета таких систем — забезпечити захист інформаційних ресурсів від атак, проникнення шкідливого коду, спроб несанкціонованого доступу та інших загроз. У разі використання IPS система не лише фіксує загрозу, а й здійснює активні дії для її нейтралізації, наприклад блокує підозрілий трафік або IP-адресу зловмисника, тим самим запобігаючи подальшому поширенню атаки.

За принципом реагування системи поділяються на два основні типи. IDS (Intrusion Detection System) працює в пасивному режимі. Вона аналізує мережевий трафік, наприклад через SPAN-порт або віддзеркалювання потоків, і виявляє аномалії

чи ознаки атак. IDS не зупиняє атаку, а лише повідомляє адміністратора про потенційну загрозу. Інформація, яку надає IDS, включає тип атаки, джерело загрози, цільові хости та час події, що дозволяє фахівцям своєчасно реагувати. Переваги IDS полягають у мінімальному впливі на продуктивність мережі та високій точності у виявленні відомих загроз. Недоліки включають те, що система не може самостійно зупинити атаку, тому ефективність реагування залежить від швидкості дій адміністратора.

IPS (Intrusion Prevention System), навпаки, працює в активному режимі. Весь трафік проходить через IPS, яка в реальному часі аналізує пакети та виявляє шкідливі або підозрілі дії. Система може автоматично блокувати окремі пакети, фільтрувати трафік або повністю заблокувати IP-адресу джерела атаки. IPS дозволяє не лише виявляти загрози, а й запобігати їхньому поширенню, що робить її критично важливою для захисту корпоративних мереж і сервісів. Переваги IPS включають активне блокування атак та можливість автоматичного реагування без втручання адміністратора. Недоліки: високе навантаження на мережу та сервер, а також ризик хибних спрацьовувань, коли легітимний трафік може бути помилково заблокований.

За місцем розташування системи також поділяються на два типи. NIDS (Network-based IDS/IPS) встановлюється на мережевий сегмента і аналізує весь трафік, що через нього проходить. Така система дозволяє виявляти атаки, націлені на будь-які пристрої сегменту, включаючи сервери, робочі станції та мережеві пристрої. NIDS ефективна для виявлення сканувань мережі, DDoS-атак, спроб віддаленого доступу та інших мережевих загроз. Переваги NIDS полягають у можливості моніторингу великого сегменту мережі, пасивній роботі без втручання в трафік та централізованому зборі даних. Недоліки включають неможливість аналізувати зашифрований трафік і обмежену ефективність при атаках, спрямованих безпосередньо на окремий хост.

HIDS (Host-based IDS/IPS) встановлюється безпосередньо на конкретний хост і моніторить внутрішні події системи. HIDS аналізує лог-файли, системні виклики,

цілісність критичних файлів, реєстраційні записи та інші показники поведінки хоста. Це дозволяє виявляти атаки, які можуть не проявлятися на рівні мережевого трафіку, такі як внутрішні модифікації системних файлів, спроби ескалації привілеїв або встановлення шкідливого ПЗ. Переваги HIDS включають детальний аналіз конкретного хоста, здатність виявляти внутрішні загрози та підтримку політик цілісності файлів. Недоліки цього є висока залежність від конфігурації та обмежений охопленням лише того хоста, на якому встановлена система.

Таким чином, комбіноване використання IDS та IPS, а також мережевих і хостових рішень дозволяє створити комплексну систему захисту, здатну не лише виявляти загрози, а й запобігати їм на різних рівнях інфраструктури.

Така інтеграція забезпечує як видимість усіх потенційних атак, так і можливість оперативного реагування, підвищуючи безпеку мережі та інформаційних ресурсів організації.

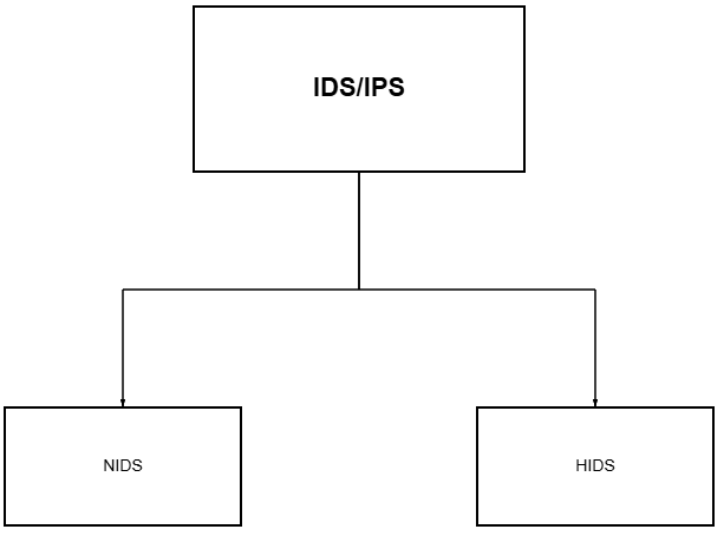

Рисунок 1.3 –Схема IDS/IPS

1.4 Методи детектування мережевих атак

Ядро будь-якої системи IDS/IPS — це модуль аналізу мережевого трафіку. Існують два фундаментальних підходи до його реалізації, які в сучасних системах часто комбінуються для підвищення ефективності захисту. Перший з них, сигнатурний аналіз, який дозволяє швидко та точно виявляти відомі атаки. Суть цього методу полягає у порівнянні мережевого трафіку з заздалегідь сформованою базою сигнатур атак. Сигнатура це своєрідний «відбиток» конкретної загрози, який може проявлятися у специфічних рядках запитів, послідовності команд чи характерних пакетах мережевого протоколу.

Приклади сигнатур включають: наявність рядка `UNION SELECT` у HTTP-запиті, що свідчить про спробу SQL-ін'єкції; звернення до `/etc/passwd` у URL, що може вказувати на спробу доступу до системних файлів; специфічний набір команд у протоколі SMB, що характерний для певних видів атак; або рядки у пакетах, які залишає сканер nmap під час перевірки мережі.

У цій роботі реалізація сигнатурного аналізу здійснюється через файл `ids_rules.yaml`, де кожне правило містить поле `payload_regex`. Це регулярний вираз, який точно описує сигнатуру конкретної атаки. Коли пакет або запит збігається з цим регулярним виразом, система фіксує подію як тривогу та за потреби може застосувати блокування або інше реагування. Такий підхід забезпечує високу точність виявлення відомих атак, із низьким рівнем хибних спрацьовувань, що робить його ефективним у повсякденному використанні та дозволяє адміністратору не пропускати критичні події.

Проте сигнатурний аналіз має й певні обмеження. Зокрема, він абсолютно неефективний проти нових (0-day) атак, для яких ще не існує сигнатур. Крім того, для підтримки актуальності та ефективності системи необхідне постійне оновлення бази правил, що потребує часу та ресурсів. Іншими словами, сигнатурний підхід добре

працює лише тоді, коли атаки вже відомі та їхні «відбитки» заздалегідь внесені до системи.

Щоб компенсувати ці недоліки, сучасні IDS/IPS часто комбінують сигнатурний аналіз з аномальним аналізом або іншими методами поведінкового виявлення. Сигнатурний аналіз надає основу для швидкого та точного виявлення відомих загроз, тоді як аномальний аналіз дозволяє системі реагувати на нові та невідомі атаки, виявляючи відхилення від нормальної поведінки мережі. Така комбінація робить систему більш універсальною, надійною та ефективною в умовах сучасних кіберзагроз.

Таким чином, сигнатурний аналіз є ключовим компонентом будь-якої IDS/IPS, забезпечує точне та швидке виявлення відомих атак, дозволяє ефективно працювати адміністраторам і є базовим елементом для подальшої інтеграції з іншими методами виявлення загроз, що робить систему комплексною та стійкою до більшості сценаріїв атак.

1.4.1 Сигнатурний аналіз

Сигнатурний метод виявлення атак є одним із найдавніших та водночас найбільш інтуїтивно зрозумілих підходів до мережевої безпеки, який багато років залишався основою класичних IDS/IPS-систем.

Його принцип ґрунтується на тому, що кожна кібератака має певний характерний відбиток сигнатуру, тобто унікальний шаблон, який дає змогу однозначно впізнати конкретну форму шкідливої активності. Якщо проводити аналогію з реальним життям, сигнатура працює як картка розшуку, у якій зафіксовані прикмети порушника, а сама система виконує роль охоронця, який порівнює кожну нову подію з уже відомими шаблонами злочинної поведінки. Коли знаходиться збіг,

IDS/IPS одразу реагує: або фіксує підозрілу подію, або, у разі роботи в режимі IPS, блокує її ще до того, як вона може вплинути на систему.

У процесі аналізу трафіку сигнатурний підхід передбачає кілька фундаментальних етапів, які відбуваються автоматично та надзвичайно швидко.

Спочатку система отримує черговий мережевий пакет, витягнутий сніфером із мережевого інтерфейсу. Далі цей пакет розкладається на структурні компоненти заголовки IP, TCP/UDP, службові поля протоколів, корисне навантаження Raw. Кожен з елементів порівнюється з великою базою сигнатур, що містить заздалегідь визначені ознаки атак: від характерних фрагментів SQL-ін'єкцій (наприклад, «UNION SELECT» або «OR 1=1») і до унікальних послідовностей байтів, які зустрічаються у певних експлойтах, ботах або шкідливих скриптах. Якщо знайдено збіг між відомою сигнатурою і вмістом пакета, система класифікує його як загрозу та застосовує відповідний механізм реагування - це логування, відправлення сповіщення або миттєве блокування IP-адреси чи TCP-сесії. Цей процес настільки оптимізований, що дозволяє аналізувати величезні обсяги трафіку в реальному часі без суттєвого навантаження на апаратні ресурси.

На практиці сигнатури можуть мати найрізноманітнішу природу. Частина з них базується на текстових шаблонах і використовується при виявленні веб-атак, таких як SQL injection, XSS чи ін'єкції команд ОС. Інші сигнатури є чисто байтовими вони містять унікальні фрагменти машинного коду, що зустрічаються у специфічних вірусах або експлойтах. Також поширені сигнатури протокольних аномалій: незвичні комбінації прапорів TCP, наприклад FIN+URG+PSH, або порушення стандартів RFC, що може свідчити про сканування портів або спроби обійти фаєрвол. До окремої категорії належать так звані поведінкові сигнатури, які описують повторювану активність, характерну для brute-force чи ботнетних запитів. Нарешті, існують сигнатури, що базуються просто на IP-адресах, доменах чи URL, пов'язаних із відомими ботнетами або C2-серверами. Усі вони, по суті, є зафіксованими шаблонами минулих атак, які система може впізнати під час наступних інцидентів.

Однак важливо враховувати, що сучасні атакувальники часто намагаються уникнути детекції шляхом модифікації своїх шкідливих інструментів. Навіть невелика зміна у послідовності байтів або додавання зайвого символу в рядок SQL-команди може обійти класичний сигнатурний фільтр. З цієї причини інколи застосовуються алгоритми «розмитого порівняння» або fuzzy matching, які дозволяють виявити схожість між атакою і вже відомою сигнатурою навіть тоді, коли вони не збігаються повністю. Такі методи підвищують гнучкість системи, але зменшують точність, збільшуючи ризик хибних спрацьовувань, тому використовуються обережно.

Ключовою перевагою сигнатурного методу є його висока точність. Якщо знайдено збіг з відомим шаблоном, імовірність помилки є мінімальною, що робить цей метод дуже надійним для критичних систем. Сигнатурні IDS/IPS майже не генерують хибних позитивів, адже реагують лише на точно описані та перевірені патерни шкідливої активності. Іншою важливою перевагою є швидкість процес пошуку збігів надзвичайно оптимізований і завдяки цьому не створює відчутного навантаження навіть у високошвидкісних мережах. Крім того, адміністрування такої системи є простим: додавання нової сигнатури фактично зводиться до оновлення бази.

Водночас сигнатурний підхід має низку критичних недоліків. Насамперед він абсолютно не здатен виявляти нові, раніше невідомі атаки. Якщо шаблон не було додано до бази, система просто не впізнає загрозу.

Це означає, що сигнатурний аналіз живе у минулому він може реагувати лише на те, що вже сталося у світі кібербезпеки. Інша вада полягає у необхідності постійного оновлення бази сигнатур: щодня з’являються нові експлойти та техніки обходу захисту, тому без активної підтримки база швидко втрачає актуальність. Крім того, цей метод дуже вразливий до модифікацій атак, навіть найменших — ситуація, якою активно користуються зловмисники, застосовуючи поліморфізм, шифрування корисного навантаження та інші техніки ухилення. І, нарешті, найбільшим структурним недоліком є те, що сигнатурна система не розуміє поведінкових

закономірностей і не може аналізувати складні багатоступеневі атаки, у яких кожен окремий пакет може виглядати законно, але загальна послідовність дій свідчить про вторгнення.

1.4.2 Аналіз аномалій

Аномальний метод виявлення вторгнень ґрунтується на фундаментальній ідеї що будь-яка комп'ютерна мережа або інформаційна система має свій характерний, стабільний режим роботи, який формується на основі повторюваних патернів активності. Якщо ця нормальна поведінка достатньо добре вивчити та зафіксувати середню інтенсивність трафіку, типові протоколи, середні розміри та частоту пакетів, характерне навантаження на сервери, звичні години пікової активності користувачів, середню кількість одночасних з'єднань та інші параметри то будь-яке значне відхилення від цього профілю вже може вказувати на потенційну атаку. Це можна порівняти з роботою охоронця, який добре знає будівлю: він може не мати повного списку всіх можливих злодіїв, але чудово бачить, коли щось відбувається не так, як зазвичай світло вмикається о другій ночі, чути незвичний шум або хтось рухається у заборонених коридорах. Так само і аномальний IDS реагує на зміну звичних патернів трафіку, навіть якщо ніколи раніше не стикався з подібною атакою.

Аномалії можуть проявлятися у різних формах.

Статистичні аномалії спостерігаються тоді, коли інтенсивність потоку даних раптово збільшується або зменшується в кілька разів, що часто свідчить про DoS/DDoS-атаки, сканування портів чи нетипову поведінку заражених хостів. Протокольні аномалії виникають, коли протокол або сервіс використовуються не у спосіб, передбачений специфікацією: наприклад, сервер, який зазвичай не ініціює вихідних з'єднань, раптом починає масово надсилати SYN-пакети - це типова ознака

командно-контрольної (C2) активності. Поведінкові аномалії стосуються лише нетипової поведінки користувачів або сервісів: масові нічні спроби входу в SSH, аномально великі завантаження файлів, підозрілі шаблони HTTP-запитів або нетипові послідовності команд. Сучасні IDS також враховують контекст: наприклад, сезонні зміни трафіку чи типові пікові навантаження.

У межах цієї роботи застосовано статистичний метод на основі ковзного вікна. Параметр `ANOMALY_WINDOW` визначає часовий інтервал, у межах якого система відстежує часові мітки пакетів для кожної IP-адреси. Методи `_update_rate` та `_check_anomaly` постійно оновлюють структуру `self.src_times`, обчислюють кількість пакетів за останній проміжок часу та порівнюють цю величину з порогом ANOMALY_THRESHOLD. Якщо протягом часу T було отримано N пакетів, система обчислює інтенсивність трафіку за формулою

$$\lambda(t) = N / T.$$

Якщо це значення перевищує нормальний профіль, фіксується подія `packet_rate_exceeded`.

Загальне правило виявлення аномалії визначається як:

$$\lambda(t) > \lambda_0 + \delta,$$

де λ_0 - це середня інтенсивність трафіку у нормальному стані,

а δ – це допустиме статистичне відхилення. У більш складних системах межа визначається за формулою

$$\lambda(t) > \lambda_0 + k \cdot \sigma,$$

де σ - це стандартне відхилення, а k - це коефіцієнт чутливості.

Класичні підходи включають:

– гаусову модель, де аномаліями вважаються значення, що виходять за межі ($\mu \pm 3\sigma$);

– метод гістограм**,** де формується емпіричний розподіл кожного параметра, а рідкісні або невідомі значення трактуються як підозрілі;

– адаптивні моделі зі згладжуванням**,** у яких базовий рівень активності постійно оновлюється за формулою.

$$\lambda_{ω}(t) = \alpha \cdot \lambda(t) + (1 - \alpha) \cdot \lambda_{ω}(t-1)$$

і це дозволяє системі підлаштовуватися до природних змін у мережевому середовищі.

Основна перевага аномального аналізу полягає у тому, що він дозволяє виявляти невідомі загрози: нові варіанти атак, 0-day експлойти, внутрішню шкідливу активність, а також складні сценарії сканування чи несанкціонованого доступу, які не мають сигнатур. Проте метод має і недоліки: високий рівень хибних спрацьовувань, складність формування коректного «нормального профілю», чутливість до сезонності та зміни контексту. Через це сучасні системи виявлення часто поєднують аномальний підхід із сигнатурним аналізом, застосовують машинне навчання (Isolation Forest, One-Class SVM, K-Means) та кореляційні правила для комплексного виявлення загроз.

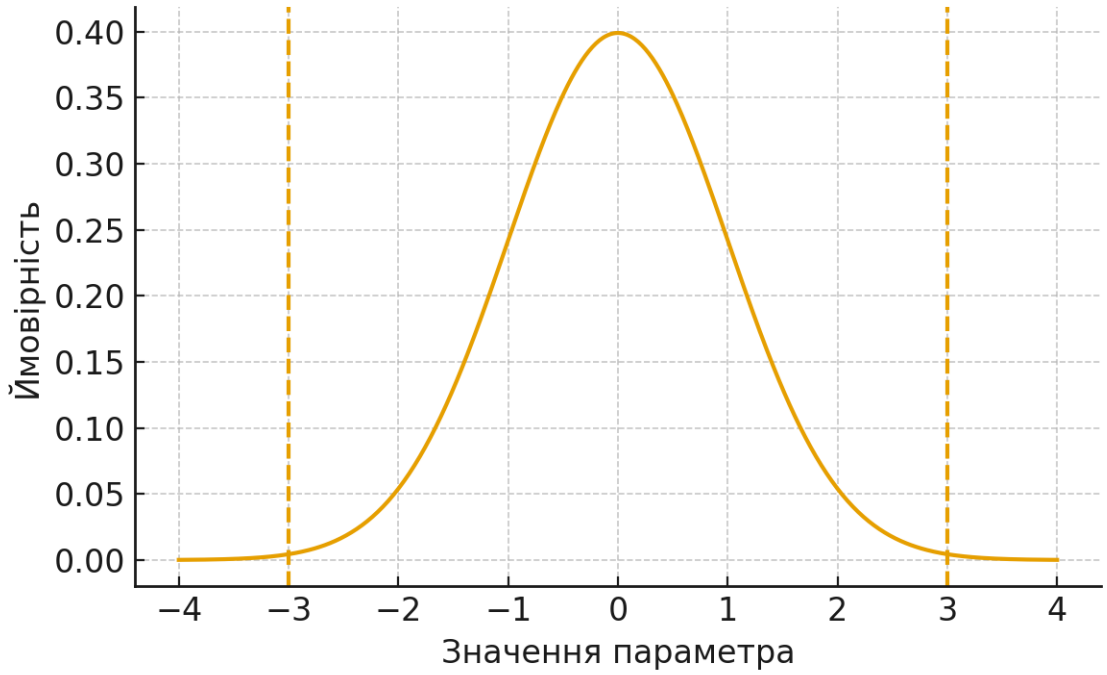

Рисунок 1.4.- Гасусів розподіл з порогами аномалій($\mu\pm$)

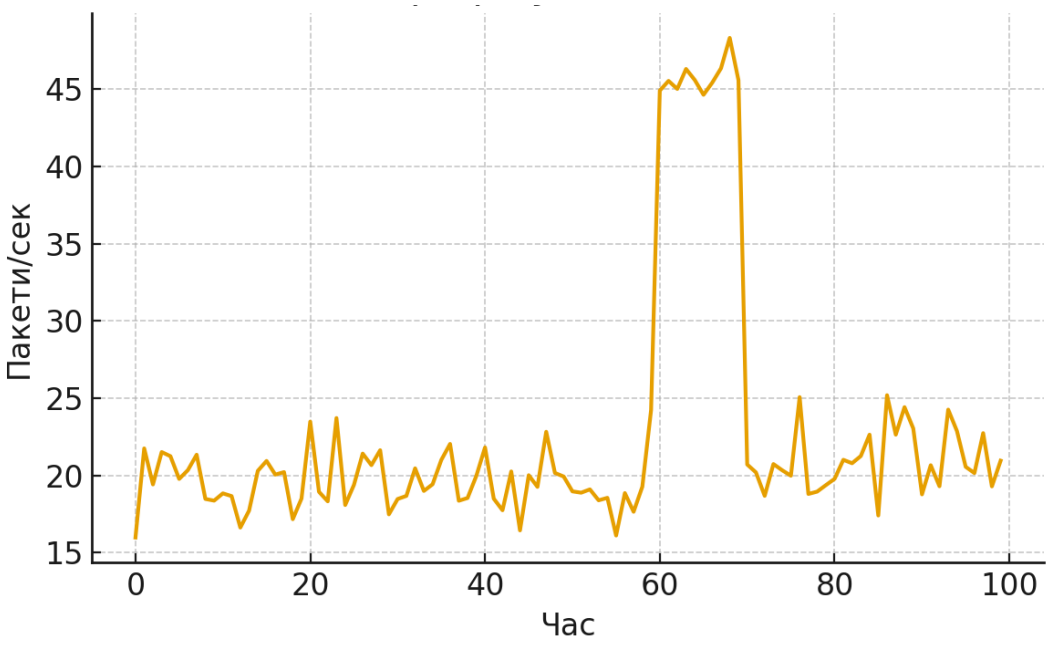

Рисунок 1.5 – Інтенсивність трафіку з аномалією

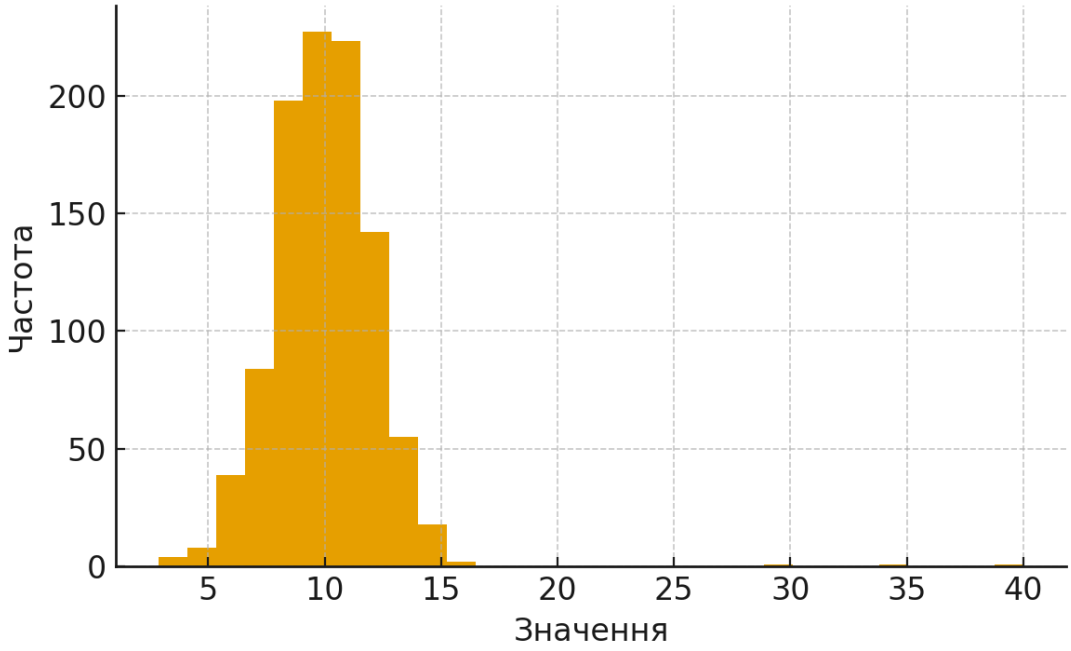

Рисунок 1.6 – Гістограма розподілу значень з рідкисними аномаліями

1.4.3 Гібридний підхід

Дана магістерська робота базується на гібридному підході до побудови системи виявлення та запобігання вторгненням, який поєднує сильні сторони як сигнатурних, так і поведінкових (аномальних) методів аналізу трафіку. Саме інтеграція цих двох підходів є ключовою науковою новизною й практичною цінністю роботи, оскільки дозволяє забезпечити високий рівень точності виявлення загроз за мінімально можливої кількості хибних спрацювань, а також значно розширити спектр атак, що можуть бути детектовані системою в реальному часі.

Сигнатурний рушій payload_regex, реалізований у системі, орієнтований на аналіз вмісту мережевих пакетів та пошук специфічних патернів, характерних для

відомих експлойтів. У рамках цієї роботи він налаштований на виявлення найбільш критичних і поширених типів атак, серед яких:

- SQL Injection – це пошук підрядків, що відповідають відомим ін'єкційним конструкціям (UNION SELECT, sleep(), ' or 1=1 тощо);
- Remote Code Execution (RCE) - це виявлення системних викликів, шкідливих команд, структури веб-експлойтів;
- EternalBlue / MS17-010 – це пошук характерних послідовностей байтів у SMB-пакетах, відомих payload-хедерів та аномалій у структурі протоколу.

Сигнатурний метод є надзвичайно ефективним для ідентифікації чітко описаних атак, що мають усталені ознаки. Проте він слабкий проти нових, модифікованих або раніше невідомих загроз саме тому його можливості підсилює другий компонент системи.

Аномальна функція packet_rate_exceeded ґрунтується на статистичному аналізі поведінки мережі та оцінці відхилень від нормального профілю трафіку.

У роботі він застосовується для виявлення:

- сканування Nmap аномальні пікові послідовності SYN-пакетів, систематичний перебір портів, нерівномірність міжпакетного інтервалу;
- DoS/DDoS-флуду перевищення порогових значень інтенсивності $\lambda(t)$, збільшення ентропії у наборі джерел, підозрілий обсяг пакетів за одиницю часу.

У теоретичному аспекті модель аномалій може бути формалізована за допомогою показників типу:

- інтенсивність потоку пакетів$\lambda(t) = N / \Delta t$;
- ентропія джерел та призначень $H = -\sum p(x) \log p(x)$;
- відхилення від середньоквадратичного значення;
- оцінка різкості приросту трафіку на основі похідної $\Delta\lambda(t)$.

Таким чином, поведінкова детекція дозволяє системі фіксувати нетипові стани мережі навіть у тих випадках, коли сигнатури експлойту не існують або коли атака виконується з використанням кастомного чи обфускованого payload.

Комбінування сигнатурного та аномального аналізу дозволило побудувати ешелоновану систему захисту, у якій кожний механізм компенсує слабкі сторони іншого. Сигнатури забезпечують точність, а аномалії широту покриття. У повному складі це дає змогу ефективно виявляти як відомі, так і нові види атак, включно з розподіленими мережевими скануваннями, DoS-флудами, складними багатоступінчастими атаками, а також спробами експлуатації критичних сервісів.

Таким чином, запропонований гібридний підхід дозволяє сформувати цілісний багаторівневий механізм контролю трафіку, який забезпечує не лише високу точність виявлення, але й адаптивність, що особливо важливо в умовах динамічного розвитку сучасних кіберзагроз. Створена система демонструє здатність ефективно реагувати на широке коло вторгнень і може бути інтегрована у корпоративні мережі як модуль реального часу для підвищення стійкості інфраструктури до атак.

1.4.4 Огляд існуючих інструментів та методологій

Для побудови ефективної та стійкої системи захисту Blue Team критично необхідно повністю розуміти інструментарій атаки, яким користується Red Team, а також типові сценарії для їх застосування. Протидія сучасним кіберзагрозам неможлива без чіткої декомпозиції логіки атакувальних дій, аналізу методологій проведення пентесту та розуміння того, як саме зловмисники виконують кожний етап атаки від первинної розвідки до повного контролю над доменною інфраструктурою. Саме тому Blue Team має мислити за принципом знати ворога краще, ніж він знає мережу, і відповідно будувати свою систему моніторингу, детекції та реагування.

У професійній спільноті існують стандартизовані методологій, що формують фундамент сучасних Red Team-операцій. До них належать OSSTMM (Open Source Security Testing Methodology Manual), NIST SP 800-115 (Technical Guide to Information Security Testing), OWASP Testing Guide та PTES (Penetration Testing Execution Standard). Вони описують повний життєвий цикл тестування на проникнення, визначають структуру атак, набір технік та необхідні артефакти, які збирає нападник.

Для Blue Team такі методології є своєрідною мапою загроз, за допомогою якої можна передбачати вектори проникнення та формувати точніші методи детекції.

Важливу роль також мають спеціалізовані дистрибутиви Linux, орієнтовані на тестування безпеки: Kali Linux, Parrot Security OS, BlackArch Linux.Це не тільки просто операційні системи але й це повноцінні платформи для проведення Red Team-операцій, оснащені сотнями попередньо встановлених інструментів, що охоплюють усі стадії Kill Chain:

- сканування портів;
- збір інформації;
- експлуатацію вразливостей;
- ескалацію привілеїв;
- крадіжку облікових даних;
- атаки на Active Directory;
- аналіз графів довіри домену;
- автоматизацію lateral movement тощо;

Саме тому системи класу IDS/IPS повинні бути здатні ідентифікувати тип патернів трафіку, які генерують ці інструменти.

До ключових інструментів Red Team, на яких варто зупинитися детальніше, належать:

- Nmap - це один із найпоширеніших мережевих сканерів, що використовується для сканування портів, визначення типів хостів, ОС, версій сервісів, наявності міжмережевих екранів та маршрутизаційних політик. Сканування портів

часто створює аномальні патерни трафіку, які повинен ідентифікувати аномальний рушій IDS. Наприклад, надмірне число SYN-запитів у короткий проміжок часу, систематичний обхід портів у порядку зростання або спадання, спроби використання нестандартних технік (SYN/FIN-сканування, NULL-сканування, XMAS-сканування). Наша система повинна формувати сигнатурні та поведінкові правила, що дозволяють фіксувати такі дії з високою точністю.

- MetasploitFramework - це універсальна платформа для створення, тестування та виконання експлойтів, модулів постексплуатації та payload-імплантів. Metasploit є стандартом у сфері пентесту, і його присутність у мережевому трафіку часто можна визначити за характерними шаблонами передачею великої кількості дрібних фрагментів даних, нетиповим набором байтів у payload, ознаками відомих експлойтів (наприклад MS17-010), характерними handshake-аномаліями. IDS/IPS повинна вміти корелювати такі аномалії з поведінкою внутрішніх вузлів, фіксувати розриви у протоколах, підозрілі зміни TTL та час між пакетами.

- Mimikatz - це класичний інструмент для екстракції облікових даних із пам’яті процесу LSASS у Windows. Хоча Mimikatz не генерує трафік напряму, його запуск майже завжди є наслідком компрометації через мережу. Тому IDS повинна виявляти попередні етапи атаки: підозрілі SMB-сесії, кейлогінг, передачу двійкових payload, використання PsExec або WMI для доставки виконуваного файлу, мережеві запити, що передують атакам Pass-the-Hash / Pass-the-Ticket.

- Responder,Hashcat,BloodHound а ці інструменти орієнтовані на атаки проти Active Directory, перехоплення хешів NTLM у мережі, аналіз привілейованих відносин, побудову графів довіри та виявлення найкоротших шляхів до доменного адміністратора. Їхня активність залишає характерні сліди: нетипові NBNS/LLMNR-пакети, multicast-трафік із підміною адрес, підозрілі граф-запити до Neo4j, спроби масового запиту атрибутів LDAP. Для Blue Team це критично важливі індикатори раннього попередження.

З боку захисту інструментами виступає розроблена в рамках цієї роботи гібридна система IDS/IPS, що була спроєктована та протестована у третьому розділі. Вона виконує функції моніторингу трафіку, аналізу аномалій, виявлення сигнатур відомих атак, а також забезпечує активне блокування загроз у реальному часі. На відміну від пасивних рішень, система не лише фіксує порушення, а й виконує негайні дії з нейтралізації атаки: блокування IP-адрес, припинення небезпечних сесій, ізоляцію вузла, протидію типам трафіку, що характерні для інструментів Red Team.

1.5 Постановка задачі дослідження.

На основі проведеного аналізу актуальності проблеми кіберзахисту, особливостей сучасної предметної області та детального вивчення існуючих рішень було сформульовано головну мету магістерської роботи. Вона полягає у підвищенні рівня безпеки корпоративних мереж шляхом створення та програмної реалізації гібридної системи виявлення й запобігання вторгненням (IDS/IPS), здатної в режимі реального часу ідентифікувати, коректно класифікувати, автоматично блокувати та наочно візуалізувати багатоетапні кібератаки.

Під час дослідження було проаналізувано можливості різних архітектур IDS/IPS сигнатурних, аномальних, гібридних та поведінкових моделей. На основі порівняння їх обмежень і переваг обґрунтовано вибір саме гібридного підходу. Така модель дозволяє поєднати високу точність сигнатурного аналізу зі здатністю аномального модуля виявляти невідомі та Zero-Day атаки, які не мають готових правил. Таким чином, система здатна реагувати не лише на шаблонні вектори атак, а й на нестандартні та раніше не зафіксовані відхилення у мережевій активності.

Для програмної реалізації було обрано стек технологій Python, оскільки ця екосистема забезпечує високу гнучкість, достатню продуктивність та величезний набір бібліотек, адаптованих під мережевий аналіз. Зокрема:

- scapy для низькорівневого перехоплення, парсингу та генерації пакетів,
- flask для створення веб-інтерфейсу моніторингу,
- subprocess для інтеграції системних команд і керування firewall-правилами,
- pyyaml, threading, logging та інші допоміжні модулі.

Було спроєктовано модульну архітектуру системи, що включає п’ять ключових підсистем:

- збор трафіку відповідає за захоплення та первинну обробку пакетів;
- аналіз містить сигнатурний та аномальний модуль, правила яких динамічно оновлюються;
- активне реагування реалізує автоматичне блокування IP-адрес із застосуванням тимчасових або постійних правил;
- сповіщення забезпечує миттєву доставку критичних інцидентів через Telegram-бота;
- візуалізації та логування веб-панель на Flask із графіками, статистикою й хронологією подій.

Окрему увагу приділено розробці сигнатурного правила, яке описується в YAML-форматі та містить регулярні вирази для виявлення атак, що мають формалізовані ознаки (наприклад, безпосередні SQLi-патерни, нестандартні HTTP-запити, специфічні послідовності TCP-флагів тощо). Паралельно з цим було реалізовано аномальний прапорець, який базується на статистичному аналізі швидкості, частоти та структури трафіку. Він дозволяє помічати підозрілу поведінку, таку як раптові піки пакетів, нетипові інтервали між запитами чи порушення нормального профілю трафіку.

Також було розроблено кросплатформовий модуль активного блокуванняМодуль веде власний стан перелік заблокованих джерел із мітками часу, що дозволяє створювати тимчасові (TTL-блокування) та постійні заборони. Це наближує систему не лише до IDS, а й до повноцінного IPS-режиму.

Для підвищення оперативності було реалізовано підсистему сповіщень через Telegram-бота, яка відправляє адміністратору інформацію про інциденти, включно з джерелом атаки, типом загрози, портом призначення та часом фіксації.

Візуалізаційний модуль на Flask формує інтерактивний дашборд, де відображаються графіки активності, кількість виявлених інцидентів, класифікація атак, топ-джерела трафіку та журнали подій у реальному часі.

Ефективність запропонованої системи буде перевірено експериментально шляхом імітації реальних атак у контрольованому середовищі. План тестування включає такі вектори, як:

- Nmap-сканування з різними режимами (SYN-scan, Xmas-scan, stealth-scan),
- SQL-ін’єкції у вигляді сформованих HTTP-запитів,
- EternalBlue (MS17-010) експлуатація критичної вразливості Windows Server 2012 R2.

Таке тестування дозволить оцінити швидкість реакції, точність спрацювання сигнатурних правил, здатність аномального модуля реагувати на відхилення та ефективність активного блокування в реальному часі.

2 ПРОЄКТУВАННЯ ТА ОБҐРУНТУВАННЯ СТЕКУ ТЕХНОЛОГІЙ IDS/IPS

В цьому розділі буде описано архітектуру системи та детально пояснено, чому було використанно саме ці технології та бібліотеки та мову програмування Python

2.1 Загальна архітектура програмного комплексу

Система була створена за модульним принципом, і цей підхід став фундаментальною особливістю всієї розробки. Замість монолітної, громіздкої архітектури, де будь-яка зміна однієї частини призводить до непередбачуваних наслідків у всій структурі, було застосовано концепцію чітких відокремлених компонентів з високою внутрішньою монолітністю та мінімальною зовнішньою залежністю. Це забезпечило можливість гнучкого масштабування, простого внесення змін та модернізації кожного модуля окремо без ризику порушення роботи всієї системи. По суті, архітектура є багатошаровою платформою, у якій кожен модуль виконує одну чітку функцію, але водночас тісно інтегрується з іншими через контрольовані точки взаємодії. Усі ключові алгоритми, моделі та процедури були реалізовані у головному класі IDS та його допоміжних функціональних класах Blocker і Telegram, які разом формують логічне ядро системи.

Першим критичним компонентом роботи став модуль збору даних, або Sniffer. Його головна функція полягає у неперервному перехопленні кожного мережевого пакета, що проходить через заданий мережевий інтерфейс. Використовуючи бібліотеку scapy та її низькорівневий механізм sniff, цей модуль фактично створює

вікно в реальний мережевий трафік. У режимі реального часу він спрацьовує на кожний вхідний пакет, передаючи його обробнику без затримок, що дозволяє системі IDS оперативно реагувати на зміни мережевої активності. Саме Sniffer забезпечує фундамент для аналізу без точного й безперервного збору даних будь-яка система виявлення вторгнень перетворюється на статичний інструмент, який нічого не бачить. Особливістю реалізації є те, що модуль працює без збереження трафіку на диск (store=False), що виключає надмірне навантаження та дозволяє системі функціонувати навіть на слабкому сервері або віртуальній машині.

Другим компонентом є модуль аналізу, де ключову роль виконує метод `_handle_packet`.

Це логічне ядро, через яке проходить кожна одиниця трафіку після перехоплення. Саме тут пакет розбирається на складові частини IP-заголовки, транспортні протоколи TCP або UDP, а також сирі дані Raw, якщо вони присутні. Метод перевіряє валідність структури пакета, узгодженість його полів, визначає тип протоколу та формує уніфіковану внутрішню модель даних, зручну для подальшого аналізу. Цей модуль виконує функцію розбору , який вирішує, куди направити пакет далі до сигнатурного або аномального детектора, чи, можливо, він не становить жодної загрози та може бути проігнорований. Важливо, що розбір пакета відбувається на глибині мережевого стека, що дозволяє IDS працювати не лише з HTTP чи DNS, а з будь-якими типами трафіку, включно з нестандартними або навіть кастомними протоколами.

Третій модуль, підсистема детекції, складається з двох незалежних, але взаємодоповнюючих підсистем сигнатурної та аномалій. Сигнатурний детектор працює як класичний механізм зіставлення з шаблонами: для кожного пакета він у циклі перевіряє відповідність набору правил із YAML-файлу. Ці правила містять регулярні вирази, специфічні ознаки атак або сигнатури характерних експлойтів, таких як SQL-ін'єкції, RCE або специфічні послідовності байтів, що відповідають відомим експлойтам на кшталт EternalBlue. Такий підхід забезпечує точне й швидке

розпізнавання загроз, які вже мають документовані характеристики та відомі в інформаційній безпеці.

Аномальний детектор, навпаки, базується на поведінковому підході. Він відстежує частоту запитів, їх динаміку, структуру та інтенсивність на основі статистичної моделі. Методи `_update_rate i _check_anomaly` формують поведінковий профіль кожної IP-адреси. Якщо від джерела спостерігається раптове зростання кількості пакетів, що перевищує адаптивні порогові значення, якщо відбуваються нетипові послідовності запитів або порушення логічного порядку протоколів система фіксує аномалію. Саме такий підхід дозволяє виявляти DoS-атаки, швидкі порт-сканування Nmap, підозрілу активність ботнетів або Zero-Day загрози, які ще не мають сигнатур. У результаті система отримує здатність не лише реагувати на вже відомі атаки, а й прогнозувати потенційно небезпечні події.

Четвертим модулем у структурі є підсистема реагування, або Response Engine. Коли один з детекторів фіксує загрозу, система не обмежується лише реєстрацією події, а переходить до активної фази протидії. Клас Blocker виконує функцію негайного блокування IP-адрес, з яких походить підозріла активність. Цей модуль підтримує кросплатформений режим роботи: на Linux він використовує iptables, а на Windows netsh advfirewall. Крім того, блокування має часовий характер після закінчення TTL-періоду заборона може бути знята, що забезпечує гнучкість та уникнення постійного блокування легітимних клієнтів, якщо спрацювання виявилося помилковим. Таким чином реалізована не лише функція IDS, але і повноцінна IPS-логіка активного реагування, що є ключовою особливістю гібридної системи.

П’ятим елементом архітектури є модуль оповіщень, який відповідає за миттєву передачу інформації про інциденти адміністратору. Клас TelegramNotifier забезпечує надсилання структурованих повідомлень у Telegram, що містять детальну інформацію про атаку IP-адресу джерела, тип загрози, протокол, порт, час, а також дію, яку було виконано системою. Ця можливість є критично важливою для оперативного

реагування: адміністратор не мусить постійно моніторити лог-файли або веб-інтерфейс система сама надсилає всі попередження в реальному часі.

Шостим і завершальним модулем є система візуалізації подій, реалізована на основі Flask-додатка, який запускається в окремому потоці за допомогою методу _start_api. Веб-інтерфейс виконує роль інформаційної панелі, де в режимі реального часу відображається статистика атак, графіки активності, журнали безпеки, список заблокованих IP-адрес, системний стан, логіка роботи детекторів та загальна динаміка подій у мережі. Дашборд дозволяє зручно спостерігати за станом інфраструктури, проводити аналіз після інцидентів та контролювати роботу IDS/IPS без необхідності взаємодії з консоллю.

2.2 Детальне обґрунтування вибору стеку технологій

Для забезпечення високої продуктивності та гнучкості і оптимальної розробки і для проєкту я обрав як основний інструмент був Python.До його переваг, я би відніс як чистий синтаксис, розвинена екосистема та швидкість для створення прототипів,що роблять його ідеальним вибором для реалізації проєкту .

Основу мережевого функціоналу становить бібліотека Scapy, яка є універсальним інструментом для аналізу мережевого трафіку. На противагу більш примітивним рішенням на кшталт pcapy, Scapy надає можливість не просто пасивного захоплення пакетів, а й їх миттєвої інтерпретації (дисекції). В архітектурі системи ця бібліотека застосовується в класі IDS (метод start): функція `sniff(prn=self._handle_packet)` безпосередньо направляє кожен перехоплений пакет на подальшу обробку. Такий підхід спрощує доступ до вкладених рівнів пакета, наприклад, через виклики `pkt.getlayer(IP)`, `pkt[TCP]` або `bytes(pkt[Raw].load)`.

Сигнатурний аналіз реалізовано з використанням стандартного модуля re для роботи з регулярними виразами в Python. Цей рушій застосовується у функції load_rules для попередньої компіляції шаблонів (re.compile) та в методі `_handle_packet` для безпосереднього пошуку збігів у мережевих даних `(r['payload_re'].search(payload))`. Критично важливим аспектом є компіляція шаблонів у байтовому форматі за допомогою `pat.encode()`, що забезпечує коректну роботу з мережевим трафіком, який представлений у вигляді байтів, а не текстових рядків.

Для ефективної обробки аномального аналізу використано вбудований модуль collections, зокрема структури defaultdict і deque. Вони забезпечують високу продуктивність і зручність у зберіганні тимчасових даних. Наприклад, `defaultdict(lambda: deque())` автоматично створює нову чергу для кожного нового IP-адресу, а deque дозволяє швидко додавати нові часові позначки `(append())` і видаляти старі `(popleft())`, реалізуючи таким чином «ковзне вікно» для аналізу частоти запитів.

Для створення вебінтерфейсу використовується Flask, легкий мікрофреймворк, який ідеально підходить для побудови простого REST API без надлишкової складності великих фреймворків, таких як Django. У проєкті Flask реалізовано в методі `_start_api`, де створюються API-ендпоінти (/api/status, /api/alerts) для передачі даних у форматі JSON, а також головна сторінка (/) для візуалізації дашборду.

Для активного реагування на загрози задіяно модулі subprocess і platform, які забезпечують взаємодію з операційною системою. Вони використовуються в класі Blocker для виконання системних команд блокування. Завдяки `platform.system()` система визначає тип ОС (Linux чи Windows), а subprocess.`check_call(cmd)` виконує відповідну команду iptables для Linux або netsh advfirewall для Windows. Це дозволяє зробити систему кросплатформовою.

Паралельна робота системи реалізована через потоки (threading) та обробку сигналів (signal). Оскільки `scapy.sniff()` блокує виконання, веб-сервер і механізм очищення блокувань запускаються в окремих потоках, що дозволяє їм працювати

одночасно з перехопленням трафіку. Модуль signal забезпечує коректну реакцію на команду завершення (Ctrl+C), плавно зупиняючи всі компоненти.

Для зберігання та передачі даних використано формати YAML і JSON. Файл ids_rules.yaml створений у YAML, оскільки цей формат легко читається людиною й дозволяє адміністратору зручно додавати нові правила. Формат JSON використовується для логів (ids_alerts.log) і API-відповідей (через jsonify), адже він є стандартом для обміну даними у вебсередовищі.

Бібліотека Requests застосовується у класі TelegramNotifier для надсилання повідомлень у Telegram через POST-запит до api.telegram.org, що дозволяє оперативно інформувати адміністратора про виявлені інциденти.

Для зручного запуску системи з командного рядка використовується модуль argparse, який надає повноцінний інтерфейс CLI. Він дозволяє передавати параметри запуску, такі як `--interface` для вибору мережевого інтерфейсу, --ips для активації системи блокування та --api для увімкнення вебдашборду.

2.3 Проєктування бази правил сигнатурного чиника

Сигнатурний рушій є настільки ефективним, наскільки якісною є його база правил. Формат ids_rules.yaml спроєктований для максимальної гнучкості

YAML - name: <Назва правила (для логів)>

```
proto: <TCP | UDP>
src_ip: <IP >
dst_ip: <IP >
src_port: <>
dst_port: <Порт призначення (опціонально)>
payload_regex: <Регулярний вираз для пошуку в байтах>
```

```
action: <alert | block ('alert')>
```

Виявлення Nmap

```
#  Block obvious nmap scanner
- name: Nmap User-Agent Scan
  proto: TCP
  dst_port: 80
  payload_regex: "Nmap Scripting Engine"
  action: alert
```

Рисунок 2.1- Правило виявлення nmap в форматі YAML

Виявлення SQL-ін'єкції

```
- name: HTTP SQLi Union Select
  proto: TCP
  dst_port: 80
  payload_regex: UNION\s+SELECT|SELECT\s+\*\s+FROM|OR\s+1=1
  action: alert
```

Рисунок 2.2- Правило виявлення SQL-ін’єкції в форматі YAML

Виявлення експлойту Brute-force

```
- name: SSH Brute-Force Probe
  proto: TCP
  dst_port: 22
  payload_regex: Failed password|authentication failure|Invalid user|SSH-
  action: alert
```

Рисунок 2.3-Правило виявлення Brute-force в форматі YAML

Виявлення експлойту MS17-010

```yaml
- name: EternalBlue Exploit Attempt (MS17-010)
  proto: TCP
  dst_port: 445
  payload_regex: "\xffSMB" # (Спрощена сигнатура, реальна є складнішою)
  action: block
```

Рисунок 2.5-Правило виявлення MS17-010 в форматі YAML

Виявлення активності Mimikatz це правило виявляє спробу передачі команд Mimikatz, наприклад, через Meterpreter або PsExec.

```yaml
- name: "Mimikatz Post-Exploitation Detected"
  proto: TCP
  payload_regex: "mimikatz|creds_all|lsadump::sam"
  action: alert
```

Рисунок 2.6-Правило виявлення Mimikatz в форматі YAML

2.4 Проєктування модуля аномалій

Модуль аномалій у системі IDS/IPS спроєктований для раннього виявлення багатоступеневих атак, зокрема агресивного порт-сканування, масових спроб встановлення з'єднання та фладингу мережевого трафіку. Основна мета алгоритму - це фіксація різкого збільшення інтенсивності пакетів від конкретного джерела, що є типовою ознакою інструментів на кшталт Nmap, Masscan, ботнет-сканерів чи модулів експлуатації, які перед виконанням атаки збирають інформацію про топологію мережі. У процесі роботи система зберігає часову активність кожного джерела, використовуючи структуру `self.src_times = defaultdict(lambda: deque())`, де для кожної IP-адреси підтримується окрема черга часових міток.

Коли надходить новий мережевий пакет, метод `_update_rate()` витягує IP-адресу відправника (наприклад, 192.168.1.100 – це потенційний атакувальник) та фіксує поточний час `ts = time.time()`, після чого додає цю часову мітку в чергу: `self.src_times[src].append(ts)`.

Далі система переходить до етапу очищення черги від застарілих записів, розраховуючи межу часу `cutoff = ts - ANOMALY_WINDOW` (наприклад, останні 10 секунд) та видаляючи всі значення, що вийшли за цей інтервал.

Це забезпечує "ковзне вікно", у межах якого враховується тільки актуальний трафік.

Після оновлення черги алгоритм викликає `_check_anomaly()`, де проводиться ключова перевірка: якщо кількість пакетів за останній інтервал `len(self.src_times[src])` перевищує поріг `ANOMALY_THRESHOLD` (наприклад 100 пакетів/10 секунд), відбувається спрацювання тригера. Важливо, що модуль передбачає захист від «шуму» - і ця подія фіксується лише один раз при перетині порога, що запобігає лавиноподібному створенню однакових логів.

Такий підхід дозволяє ефективно розпізнавати атаки, де зловмисник надсилає сотні пакетів за лічені секунди.

Наприклад, під час запуску Nmap з режимами SYN-scan або ping-sweep створюється висока щільність мережевих запитів, що негайно перетинає поріг аномалії.

У режимі IPS система миттєво блокує джерело трафіку ще до того, як атакувальник встигне завершити сканування та перейти до пошуку вразливостей, забезпечуючи превентивний захист мережевої інфраструктури.

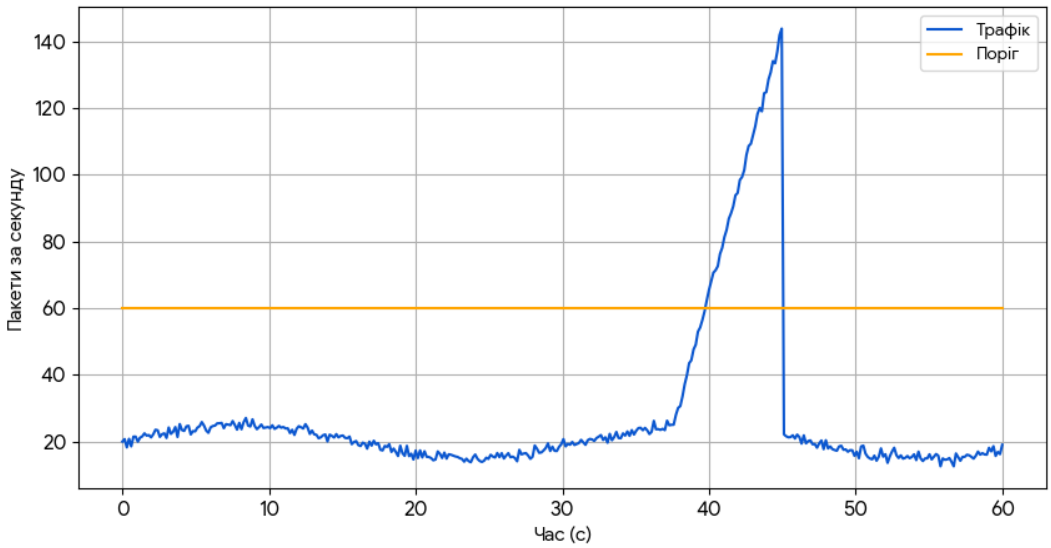

Рисунок 2.4.1- Інтесивність мережевого трафіку з аномальним сплеском

2.5 Проєктування модуля реагування (Blocker)

Модуль Blocker у цій системі відіграє ключову роль, оскільки саме він реалізує справжню логіку IPS – активне блокування джерел небезпечного трафіку. Якщо сигнатурні або аномальні механізми лише «попереджають» про загрозу, то Blocker фактично взаємодіє з мережевим стеком і змінює поведінку системи в реальному часі. Його роботу можна порівняти з охоронцем, який не просто бачить порушення, а й відчиняє або зачиняє двері для конкретних відвідувачів.

Основою модуля є словник `self.blocked`, де зберігаються IP-адреси, що були заблоковані, а також точний час, коли це блокування має завершитися.

Такий підхід дозволяє уникнути перманентних банів і водночас точно контролювати тривалість «карантину». Наприклад, якщо адреса потрапила під підозру через Nmap-сканування, їй може бути заборонено доступ на кілька хвилин і цього достатньо, аби зірвати атаку, але не завдати шкоди звичайним користувачам.

Коли система фіксує небезпечну активність, у гру вступає метод `block_ip()`. Він

працює максимально обережно та розумно: спочатку перевіряє, чи цей IP вже блокується. Якщо так, система не створює нове правило – і це замість цього продовжує час блокування. Це дуже важливо, бо без такого механізму iptables міг би захаращуватися десятками однакових рядків.

Уявимо затяжний флуд: 100 аномальних пакетів і в таблиці 100 однакових DROP-правил. Blocker цього не допускає.

Якщо IP вперше потрапляє під санкції, викликається `_apply_block(ip)`. Саме цей метод робить «важку роботу» і виконує команду:

```
iptables -I INPUT -s <ip> -j DROP
```

Правило вставляється саме на початок ланцюга INPUT, щоб DROP спрацьовував до будь-яких інших дозволяючих політик. Це дозволяє миттєво перекрити шкідливий трафік, навіть якщо атакуючий ще продовжує надсилати пакети.

Після цього системі залишається лише зафіксувати, коли це блокування має бути зняте. Поточний час плюс `BLOCK_DURATION` що дає `unblock_time`, і ця пара додається у словник `self.blocked`. Таким чином, кожен IP отримує свій таймер.

Однак заблокувати - це тільки половина справи. Щоб система працювала ефективно і не перетворювалася з часом на «чорну діру», потрібний механізм автоматичного очищення. Саме для цього створено окремий фоновий процес `_block_reaper()`. Він працює у вигляді daemon-потоку, непомітно для користувача, і кожні 5 секунд переглядає всі активні блокування. Якщо час певного IP закінчився, reaper делікатно видаляє його правило: `iptables -D INPUT -s <ip> -j DROP` і прибирає запис зі словника. Так система повертає доступ тим, хто вже «відсидів» своє.

Такий підхід забезпечує здорову циклічність і чистоту системи: вона не накопичує старі правила, не блокує людей назавжди, не створює хаосу у таблицях маршрутизації і водночас залишається достатньо суворою, щоб припинити атаку ще на ранній стадії.

Фактично Blocker поводиться як добре продуманий «шлюз безпеки»: він швидко реагує, але не карає назавжди; працює ефективно, але не агресивно; виконує

свою роботу непомітно, але системно та стабільно.

Саме через це модуль є обов’язковою частиною будь-якої IDS/IPS, яка претендує на практичне використання, а не лише на демонстрацію теорії.

2.6 Проєктування модуля візуалізації (Flask Dashboard)

У сучасних системах інформаційної безпеки адміністратори часто змушені працювати з традиційними логами, які відображаються на чорному екрані консолі. Такий підхід є не лише незручним, а й малоефективним, адже складно швидко зрозуміти стан системи, визначити, які події потребують негайної уваги, або відслідкувати поточні атаки та блокування користувачів. Для вирішення вказаної проблеми, в нашій системі реалізований спеціальний модуль візуалізації під назвою _start_api, який перетворює сирі дані та логи у зручний, інтерактивний та наочний дашборд. Цей модуль дозволяє адміністратору в режимі реального часу відстежувати стан системи, бачити поточні атаки та контролювати блокування IP-адрес, фактично створюючи мініатюрний Security Operations Center (SoC) без необхідності додаткового програмного забезпечення.

Серверна частина системи реалізована на Flask і запускає веб-сервер на порту API_PORT (за замовчуванням 5001). Основна HTML-сторінка дашборду віддається через маршрут `@app.route('/')`, який повертає `HTML_TEMPLATE`. Цей шаблон містить структуру інтерфейсу та вбудований JavaScript, який забезпечує динамічне оновлення даних без перезавантаження сторінки. Для отримання поточної інформації про стан системи використовується маршрут `@app.route('/api/status')`, який повертає JSON-об'єкт із ключовими параметрами: кількістю активних правил, статусом IPS та списком заблокованих IP-адрес, отриманих з `self.blocker.list_blocked()`. Таким

чином адміністратор може бачити не лише теоретичний стан системи, але й актуальну інформацію про те, які IP наразі заблоковані і скільки правил застосовано.

Окремий маршрут `@app.route('/api/alerts')` повертає JSON із останніми 200 тривогами, які зберігаються у списку `self.alerts`. Кожна тривога містить детальну інформацію про подію: IP-адресу джерела, протокол, порт, час надходження пакета та тип порушення. Така деталізація дозволяє адміністратору відразу оцінити серйозність атаки та прийняти рішення про подальші дії. Таким чином бекенд виконує роль постійно оновлюваного джерела даних, яке надає актуальну інформацію про стан системи, атаки та блокування.

Клієнтська частина, реалізована через HTML та JavaScript, дозволяє відображати всі ці дані у вигляді інтерактивних таблиць і панелей. Основна функція `fetchData()` виконується асинхронно і запускається кожні 5 секунд за допомогою `setInterval(fetchData, 5000)`. Вона робить запити до маршрутів `/api/status` та `/api/alerts` за допомогою `fetch()`. Після отримання JSON-відповіді дані обробляються та динамічно оновлюють DOM. Наприклад, таблиця тривог оновлюється через конструкцію `document.getElementById('alerts-table').innerHTML = renderAlerts(data.alerts)`, а список заблокованих IP відображається на окремій панелі. Крім того, інтерфейс може додатково виділяти критичні тривоги кольором, сортувати події за часом або типом атаки, а також надавати можливість адміністратору вручну розблокувати IP без перезавантаження сторінки.

Такий підхід забезпечує кілька важливих переваг. По-перше, адміністратору не потрібно вручну переглядати логи, що значно економить час та знижує ймовірність пропуску важливої події. По-друге, дані оновлюються в реальному часі, що дозволяє швидко реагувати на загрози. По-третє, інформація подається наочно у вигляді таблиць і панелей, що полегшує аналіз і розуміння стану системи навіть непрофесіоналам. По-четверте, дашборд інтегрує інформацію про правила, статус IPS, активні блокування та останні тривоги, створюючи повноцінний міні-SoC у межах одного веб-інтерфейсу.

В результаті адміністратор отримує повний огляд ситуації в мережі: він може миттєво бачити, хто заблокований, які атаки відбуваються, які правила застосовуються, а також оцінювати критичність подій. Це значно підвищує ефективність реагування, дозволяє знизити ризик хибних спрацювань та підтримувати високий рівень безпеки мережі. Така інтеграція бекенду та фронтенду створює єдину екосистему, де дані збираються, обробляються та візуалізуються в зручному форматі, що робить адміністрування системи прозорим, швидким і наочним.

3 ПРОГРАМНА РЕАЛІЗАЦІЯ ТА ЕКСПЕРИМЕНТАЛЬНЕ ТЕСТУВАННЯ ГІБРИДНОЇ IDS/IPS

Цей розділ присвячений практичній реалізації програмного комплексу, спроєктованого в Розділі 2. На попередньому етапі була сформована архітектура системи, визначені логічні модулі та створений експериментальний прототип, необхідний для перевірки працездатності та ефективності запропонованого підходу. У цьому розділі я переходжу від теоретичних концепцій до реального програмного коду, демонструючи, як саме була реалізована система виявлення та запобігання вторгненням на Python.

Основною метою є показати, що розроблений комплекс не є просто набором окремих алгоритмів, а працює як узгоджена, готова до використання система, здатна виявляти та блокувати реалістичні вектори атак, які були описані в Розділі 1. Крім того, здійснюється порівняння роботи програмного комплексу з можливостями комерційних рішень, що дозволяє оцінити його життєздатність у практичних сценаріях.

У даному розділі я описую структуру коду, логіку взаємодії модулів, принципи обробки трафіку и механізми ухвалення рішень. Також розглядається процес формування експериментального білду: як налаштовувалося тестове середовище, які пакети використовувалися, які параметри були обрані для симуляції атак та які сценарії тестування були проведені.

Окрема увага приділяється саме практичній частині — запуску системи в режимі реального часу, тестуванню аномальних, сигнатурних і гібридних методів, оцінці швидкодії, кількості хибних спрацьовувань та коректності блокування зловмисного трафіку.

Результати симуляцій подаються у вигляді логів, графіків, таблиць і порівняльних висновків.

Таким чином, цей розділ демонструє повний цикл переходу від теоретичної архітектури до робочого прототипу, підтверджує валідність обраних методів та показує, що створена система може використовуватися як реальна альтернатива базовим комерційним IDS/IPS-рішення

```
5372 INFO: Running Analysis Analysis-00.toc
5372 INFO: Target bytecode optimization level: 0
5372 INFO: Initializing module dependency graph...
5661 INFO: Initializing module graph hook caches...
5722 INFO: Analyzing modules for base_library.zip ...
8262 INFO: Processing standard module hook 'hook-heapq.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks'
8622 INFO: Processing standard module hook 'hook-encodings.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks'
13663 INFO: Processing standard module hook 'hook-pickle.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks'
26828 INFO: Caching module dependency graph...
26872 INFO: Analyzing D:\Диплом\GRMCENXVOL_\main.py
26984 INFO: Processing standard module hook 'hook-platform.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks'
28209 INFO: Processing standard module hook 'hook-urllib3.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\_pyinstaller_hooks_contrib\\stdhooks'
29026 INFO: Processing pre-safe-import-module hook 'hook-typing_extensions.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks\\pre_safe_import_modu
le'
29026 INFO: SetuptoolsInfo: initializing cached setuptools info...
41826 INFO: Processing standard module hook 'hook-charset_normalizer.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\_pyinstaller_hooks_contrib\\stdhooks'
42381 INFO: Processing standard module hook 'hook-certifi.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\_pyinstaller_hooks_contrib\\stdhooks'
44197 INFO: Processing standard module hook 'hook-difflib.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks'
44702 INFO: Processing standard module hook 'hook-_ctypes.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks'
46885 INFO: Processing standard module hook 'hook-multiprocessing.util.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks'
47193 INFO: Processing standard module hook 'hook-xml.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks'
54584 INFO: Processing standard module hook 'hook-scapy.layers.all.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks'
57739 INFO: Processing standard module hook 'hook-sysconfig.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks'
60417 INFO: Processing standard module hook 'hook-jinja2.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\_pyinstaller_hooks_contrib\\stdhooks'
62549 INFO: Processing pre-safe-import-module hook 'hook-importlib_metadata.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks\\pre_safe_import_mod
ule'
62557 INFO: Processing module hooks (post-graph stage)...
67613 INFO: Performing binary vs. data reclassification (3 entries)
67726 INFO: Looking for ctypes DLLs
67758 WARNING: Library wpcap.dll required via ctypes not found
68007 INFO: Analyzing run-time hooks ...
68023 INFO: Including run-time hook 'pyi_rth_inspect.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks\\rthooks'
68023 INFO: Including run-time hook 'pyi_rth_multiprocessing.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks\\rthooks'
68023 INFO: Including run-time hook 'pyi_rth_pkgutil.py' from 'C:\\Users\\Oksana\\AppData\\Local\\Programs\\Python\\Python311\\Lib\\site-packages\\PyInstaller\\hooks\\rthooks'
68039 INFO: Creating base_library.zip...
68080 INFO: Looking for dynamic libraries
WARNING: No libpcap provider available ! pcap won't be used
71676 INFO: Extra DLL search directories (AddDllDirectory): []
71676 INFO: Extra DLL search directories (PATH): []
74180 INFO: Warnings written to D:\Диплом\GRMCENXVOL_\build\ids_dashboard\warn-ids_dashboard.txt
74346 INFO: Graph cross-reference written to D:\Диплом\GRMCENXVOL_\build\ids_dashboard\xref-ids_dashboard.html
74393 INFO: checking PYZ
74393 INFO: Building PYZ because PYZ-00.toc is non existent
74393 INFO: Building PYZ (ZlibArchive) D:\Диплом\GRMCENXVOL_\build\ids_dashboard\PYZ-00.pyz
75279 INFO: Building PYZ (ZlibArchive) D:\Диплом\GRMCENXVOL_\build\ids_dashboard\PYZ-00.pyz completed successfully.
75526 INFO: checking PKG
75527 INFO: Building PKG because PKG-00.toc is non existent
75527 INFO: Building PKG (CArchive) ids_dashboard.pkg
79090 INFO: Building PKG (CArchive) ids_dashboard.pkg completed successfully.
79106 INFO: Bootloader C:\Users\Oksana\AppData\Local\Programs\Python\Python311\Lib\site-packages\PyInstaller\bootloader\Windows-64bit-intel\run.exe
79106 INFO: checking EXE
```

Збірка проєкту Рисунок 3.1

3.1Огляд програмної реалізації та структури коду

Програмний комплекс реалізовано як єдиний Python-файл, у якому зосереджена вся основна логіка: обробка трафіку, класифікація подій, механізми блокування, логування та запуск допоміжних сервісів. Такий підхід робить розгортання системи максимально простим: достатньо запустити один файл і система автоматично завантажує всі необхідні модулі, ініціалізує правила та переходить у режим моніторингу. Для роботи використовуються як стандартні Python-бібліотеки, так і сторонні пакети, що були обґрунтовані в Розділі 2.2.

Управління запуском здійснюється через модуль argparse, який формує зручний CLI-інтерфейс. Завдяки цьому користувач може гнучко налаштовувати роботу комплексу через набір прапорців:

- interface (-i) він вказує мережевий інтерфейс (наприклад, *eth0*), з якого система буде знімати трафік. Це критичний параметр для функції `scapy.sniff()`, адже саме через нього IDS отримує доступ до реальних пакетів.
- ips активує режим IPS. Якщо цей прапорець не передано, система працює як звичайна IDS: фіксує інциденти, але не виконує активних дій з блокування.
- api запускає веб-дашборд на Flask, який дозволяє переглядати статистику в реальному часі, аналізувати логи, перевіряти статус активних блокувань та взаємодіяти з системою безпосередньо через браузер.

Перед запуском системи виконується перевірка середовища. Програма оцінює наявність критично важливих бібліотек (`scapy`, `flask`) та перевіряє права доступу, оскільки для захоплення трафіку потрібні привілеї адміністратора. Якщо виникає PermissionError або відсутній один із модулів, система коректно завершує роботу та виводить пояснювальне повідомлення. Такий підхід дозволяє уникнути неконтрольованих помилок і забезпечує стабільність роботи при запуску.

Загалом практична реалізація зводиться до того, щоб перетворити теоретичну модель, побудовану в попередніх розділах, на робочу систему, яка запускається з командного рядка, контролює мережевий інтерфейс у реальному часі та забезпечує як пасивне виявлення загроз, так і активне блокування трафіку.

У наступних підрозділах розглядається структура коду, логіка взаємодії між модулями, процес формування експериментального білду, тестове середовище та результати симуляції атак.

3.1.1 Клас гіпервізору

Клас IDS є центральним ядром усієї системи, оскільки саме він координує роботу всіх компонентів і забезпечує логіку виявлення та реагування на інциденти. У його конструкторі `__init__` здійснюється повна ініціалізація внутрішніх модулів. Передусім система завантажує сигнатурні правила з файлу ids_rules.yaml за допомогою методу `load_rules()`. Це дає можливість централізовано керувати сигнатурами, змінювати їх без редагування коду і підтримувати структуру правил у наочному вигляді.

Далі, якщо система запущена в режимі IPS тобто параметр enable_ips = True, створюється екземпляр класу `Blocker`, який відповідає за активне блокування IP-адрес у разі виявлення загроз. Паралельно запускається модуль сповіщень `TelegramNotifier`, що забезпечує можливість оперативно передавати критичні алерти адміністратору. Для аномалій аналізу трафіку створюється спеціальна структура `self.src_times`, побудована на основі `defaultdict` і `deque`. Вона використовується для фіксації часових міток пакетів від кожного джерела, що дозволяє відстежувати частоту звернень і виявляти аномальні піки активності. Також формується буфер

`self.alerts`, який акумулює згенеровані сповіщення для подальшого виводу через веб-інтерфейс або логування.

Перехід системи в робочий режим здійснюється методом `start()`, який запускає повний життєвий цикл IDS/IPS. Якщо система працює як IPS, створюється окремий фоновий потік, що виконує метод `_block_reaper()` а він періодично перевіряє стан заблокованих IP-адрес і знімає блокування після закінчення їх часу дії. Якщо ж увімкнено веб-API, додатковий потік запускає функцію `_start_api()`, що відповідає за роботу Flask-дашборда. Цей інтерфейс дозволяє в реальному часі переглядати статистику, список заблокованих адрес, лог подій та загальний стан системи.

Основний потік програми запускає сніфер `scapy.sniff()` , який безпосередньо перехоплює пакети на вказаному мережевому інтерфейсі. Для кожного пакета викликається функція-callback `self._handle_packet`, яка здійснює аналіз вмісту, порівняння з сигнатурами, оновлення статистичних моделей та прийняття рішень про реагування. Завдяки такому підходу система працює в режимі реального часу, одночасно виконуючи кілька паралельних процесів та аналіз трафіку, обслуговування API, очищення блокувань та генерацію алертів.

У результаті клас IDS виступає як координуючий центр усієї системи, поєднуючи логіку моніторингу, виявлення, аналізу та активного реагування в єдиний цілісний програмний комплекс.

3.1.2 Модуль IPS.

Клас, що відповідає за активне реагування, реалізує повну логіку роботи підсистеми IPS і забезпечує автоматичне блокування зловмисної активності в режимі реального часу. У його конструкторі закладено механізм крос-платформової роботи: система самостійно визначає операційну систему за допомогою

platform.system().lower() і, залежно від результату, вибирає відповідний спосіб блокування мережевих підключень. Для Linux використовується утиліта iptables, тоді як у Windows це netsh advfirewall. Такий підхід дозволяє запускати систему на будь-якій популярній платформі без необхідності змінювати код.

Клас працює у станоутримувальному режимі (stateful) і зберігає внутрішній стан заблокованих адрес. Для цього використовується словник self.blocked, де ключами є IP-адреси, а значеннями час, коли дія тимчасового блокування повинна завершитися. Така модель дозволяє системі гнучко керувати блокуваннями й контролювати їхній життєвий цикл.

Основну логіку накладання блоків реалізує метод block_ip(). Якщо IP-адреса вже була заблокована раніше, система не дублює правила у брандмауері, а лише подовжує термін блокування, оновлюючи відповідну часову мітку. Якщо ж IP з'являється у списку вперше, викликається внутрішній метод _apply_block(ip), який формує команду для запровадження блокувального правила. Він виконує системні інструкції через subprocess.check_call() наприклад, команду iptables -I INPUT ... у Linux або netsh advfirewall firewall add rule ... у Windows.

Аналогічно метод _remove_block(ip) відповідає за видалення блокувального правила, використовуючи системні команди для очищення брандмауера.

Ключовим елементом цього класу є механізм автоматичного контролю тривалості блокування. У фоновому режимі працює метод _block_reaper, який кожні кілька секунд перевіряє словник self.blocked і визначає, чи настав час зняти певне обмеження. Якщо час розблокування минув, система викликає _remove_block(ip) і видаляє IP зі списку активних блокувань. Уся процедура виконується повністю автоматично, без будь-якого втручання адміністратора.

Такий підхід забезпечує системі не лише можливість своєчасно реагувати на підозрілу активність, але й самостійно підтримувати актуальний стан блокувань, запобігаючи перевантаженню брандмауера зайвими правилами та зберігаючи коректність роботи мережі. В результаті цей клас виступає важливим компонентом

IPS-підсистеми, забезпечуючи ефективне, контрольоване та автономне блокування зловмисного трафіку.

3.1.3 Клас TelegramNotifier - модуль сповіщень

Модуль сповіщень у системі відіграє роль свого роду нервова система, яка передає всі критичні сигнали адміністратору. Цей клас спеціально створений для того, щоб будь-яка важлива подія такий як підозрілий пакет, ознаки сканування портів, сплеск трафіку чи зафіксований вектор атаки відразу доходила до людини, яка відповідає за безпеку мережі.

Для того,щоб зробити це правильно і безпечно, клас не містить у собі жодних секретних даних. Усі конфіденційні параметри, такі як токен Telegram-бота та ID чату, де повинні з’являтися сповіщення, зчитуються зі змінних оточення через `os.getenv()`. Такий підхід створює стандарт у сфері кібербезпеки. Він дозволяє уникнути поширених помилок, коли ключі випадково потрапляють у GitHub, у резервні копії або в логи, після чого їх можуть перехопити сторонні особи. Це робить систему більш захищеною та професійною.

Коли IDS фіксує аномалію або атаку, вона викликає метод `send()`. Усередині нього формується коректний POST-запит і надсилається через бібліотеку requests безпосередньо до Telegram API. За секунду-дві адміністратор отримує повідомлення в чаті: що сталося, коли, з якого IP і який модуль це виявив. Фактично Telegram тут виступає зручним каналом оперативного реагування є аналогом систем моніторингу великих компаній, але у компактній та легко інтегрованій формі.

Якщо Telegram API відповідає помилкою наприклад, через неправильний токен, проблеми з інтернетом або некоректно сформоване повідомлення система може зберегти інформацію про збій у логах. Це дозволяє швидко діагностувати проблеми, і

адміністратор завжди розумітиме, чи повідомлення було успішно доставлене, чи «загубилося» d дорозі.

Цей клас також дає можливість будувати більш гнучку та інформативну систему сповіщень.

Наприклад, у майбутньому можна додати:

- автоматичну відправку log-файлів,
- прикріплення графіків або статистики по трафіку,
- інтерактивні кнопки (наприклад, «Заблокувати IP», «Дозволити», «Показати деталі»),
- мультиканальність підтримку Slack, Discord, Email або навіть SMS.

Важливо, що цей модуль працює абсолютно автономно. Він не залежить від логіки IPS чи процесу sniffing, не втручається у роботу фаєрвола, не блокує потоки. Його єдине завдання — прийняти текст повідомлення, красиво його оформити і гарантовано доправити до того, хто відповідає за захист системи.

По суті, він є своєрідним містком між «машиною» та «людиною», механізмом, який робить систему IDS/IPS не просто технічною а є і зручною, зрозумілою і ефективною в реальному використанні. Саме завдяки цьому класу адміністратор завжди в курсі того, що відбувається в мережі, навіть якщо його немає біля комп’ютера, і може швидко реагувати на загрози перш ніж вони встигнуть завдати шкоди.

3.1.4 Модуль візуалізації

Даний модуль реалізує веб-дашборд і REST API для системи виявлення та запобігання вторгнень (IDS/IPS) та є ключовим інструментом для моніторингу її роботи в реальному часі. Він дозволяє користувачу не лише переглядати статистику роботи системи, але й отримувати детальну інформацію про останні події та активні загрози без необхідності роботи з логами вручну. Бекенд побудований на Flask і запускається в окремому потоці через функцію _start_api, що забезпечує його паралельну роботу разом із основною логікою IDS/IPS, не блокуючи обробку мережевих пакетів чи аномалій.

Основна функціональність бекенду включає три ключові ендпоінти. Перший головна сторінка, яка повертає статичну HTML-сторінку (HTML_TEMPLATE), що слугує дашбордом для відображення стану системи, алертів і списку заблокованих IP. Другий ендпоінт /api/status, який повертає JSON із поточною статистикою роботи системи: кількість завантажених правил IDS/IPS (rules_loaded), кількість зафіксованих алертів (alerts), стан активності IPS (ips_enabled) та список IP-адрес, які в даний момент заблоковані (blocked_ips). Третій ендпоінт /api/alerts, який повертає JSON із останніми 200 алертами з буфера (self.alerts), дозволяючи користувачу відслідковувати нові загрози та події в системі в режимі реального часу.

Фронтенд реалізований як інтерактивна HTML-сторінка із вбудованим JavaScript, який забезпечує асинхронне оновлення даних. Скрипт на стороні клієнта використовує функцію setInterval(fetchData, 5000) для регулярного опитування серверних ендпоінтів кожні 5 секунд.

Після отримання даних у форматі JSON сторінка динамічно оновлює таблиці з алертами та інформацією про стан системи за допомогою команд типу document.getElementById('alerts-table').innerHTML = ... Це створює ефект моніторингу в режимі реального часу: користувач одразу бачить нові алерти, зміни у статусі IPS та

оновлений список заблокованих IP, що значно підвищує зручність і ефективність роботи з системою.

Модуль також забезпечує легку інтеграцію з іншими компонентами IDS/IPS, оскільки всі дані централізовано зберігаються та надаються через REST API. Це дозволяє додавати нові функціональні можливості, такі як графіки трафіку, гістограми алертів, сортування та фільтри, без необхідності змінювати основну архітектуру. Крім того, завдяки асинхронній природі оновлення даних та невеликому навантаженню на сервер, модуль може ефективно працювати навіть у мережах із високим потоком пакетів та великою кількістю алертів.

Модуль також підтримує масштабованість: за необхідності можна додати розширені функції аналітики, інтеграцію з іншими системами SIEM, автоматичне сповіщення адміністраторів через електронну пошту або Telegram, а також ведення історії алертів для подальшого аналізу. Таким чином, цей модуль забезпечує повноцінний інструмент моніторингу та контролю для будь-якої IDS/IPS-системи, підвищуючи її ефективність та зручність використання.

3.2 Деталізація реалізації ключових цілей аналізу

Ефективність системи відзначається поєднанням двох основних механізмів аналізу сигнатур та аномалій, які працюють одночасно та автоматично при надходженні кожного нового мережевого пакета. Метод `_handle_packet()` фактично є мозком системи, оскільки саме тут приймаються рішення про те, чи є пакет безпечним, підозрілим або потенційно шкідливим, і як на нього реагувати.

На першому етапі метод отримує пакет у вигляді Raw-даних і приводить його до зручного для аналізу формату. З пакета витягуються ключові атрибути: IP-адреси джерела та призначення, порти, тип протоколу та інші базові характеристики. Це

дозволяє проводити точний аналіз трафіку та швидко ідентифікувати потенційні загрози без зайвого навантаження на систему. Якщо пакет містить корисне навантаження у вигляді Raw, воно також витягується для подальшого порівняння із сигнатурними шаблонами.

Далі пакет надходить у сигнатурний рушій, який відповідає за виявлення відомих атак. Система перевіряє, чи не входить IP-адреса до чорного списку, чи не повторюється характерна послідовність пакетів, чи немає типових ознак порт-сканування або DoS-запиту. У разі виявлення збігу пакет обробляється миттєво: система створює алерт, записує інцидент у журнал, відправляє сповіщення через Telegram і, якщо активовано режим IPS, блокує джерело трафіку за допомогою модуля Blocker. Таким чином, сигнатурний механізм забезпечує швидку реакцію на вже відомі загрози та підтверджує, що стандартні атаки не пройдуть непоміченими.

Після цього пакет переходить до аналізу аномалій, де застосовується принципово інший підхід. Тут система не орієнтується на відомі шаблони, а оцінює поведінку пакета у контексті нормальної активності мережі. Підозрілими можуть стати пакети з нестандартним розміром, висока частота запитів, велика кількість одночасних з’єднань з одного джерела або інші нетипові показники. Саме завдяки аномального рушію IDS/IPS може виявляти нові, приховані або складні атаки, включаючи zero-day експлойти та ботнет-атаки, яких ще немає у сигнатурних базах. Крім того, аномальний модуль підтримує механізм дедуплікації алертів, реагуючи лише на критичні перевищення порогу активності, що запобігає спаму повідомлень і знижує навантаження на систему.

На завершальному етапі `_handle_packet()` поєднує результати обох перевірок та приймає комплексне рішення. Якщо пакет безпечний, він ігнорується, а статистика оновлюється для майбутніх оцінок нормальної активності. При виявленні підозрілих або відомих загроз система одночасно записує інцидент у журнал, відображає алерт на веб-дашборді, відправляє сповіщення через Telegram і за необхідності виконує блокування джерела трафіку.

Таким чином, `_handle_packet()` реалізує багаторівневий та комплексний підхід до захисту: сигнатурний механізм відповідає за відомі атаки, а мезанізм аналізу — за нові та нестандартні. Спільна робота обох систем дозволяє не лише швидко реагувати на вже відомі загрози, а й ефективно виявляти складні, приховані або нетипові вектори атак. Це забезпечує високу гнучкість і надійність системи, дозволяє підтримувати її роботу в умовах сучасних корпоративних мереж із великим потоком трафіку, а також гарантує, що навіть нові загрози не залишаться непоміченими.

Додатково система веде статистику пакетів та активності хостів у реальному часі, що дозволяє проводити детальний аналіз атак після їхнього завершення, оцінювати ефективність реакції та оптимізувати порогові параметри аномального рушія для підвищення точності виявлення без підвищення кількості хибних спрацьовувань. Такий підхід робить IDS/IPS не просто інструментом моніторингу, а повноцінним активним захисним механізмом для сучасних мереж.

3.2.1 Конвеєр обробки пакетів (_handle_packet)

Клас `_handle_packet()` є центральним елементом усієї системи IDS/IPS фактично її серцем, яке реагує на кожен перехоплений scapy пакет у режимі реального часу. Його робота починається з того, що система виконує диссекцію пакета: із вхідного об’єкта `pkt` витягується IP-шар (`ip = pkt.getlayer(IP)`). Якщо пакет не містить IP-рівня, він одразу відкидається як такий, що не несе цінності з точки зору безпеки. Такий фільтр дозволяє зменшити навантаження та зосередитись виключно на трафіку, який може бути потенційно шкідливим. Після цього відбувається вилучення основних атрибутів джерельної і цільової IP-адреси, портів, типу протоколу та інших метаданих, які формують паспорт кожного пакета.

Далі система перевіряє, чи не містить пакет корисного навантаження у вигляді `Raw`. Якщо так, воно витягується у вигляді байтової послідовності (`payload = bytes(pkt[Raw].load)`). Це важливий момент, адже сигнатурний рушій працює саме з вмістом пакета, намагаючись розпізнати патерни типових атак: фрагменти exploit-команд, ознаки сканування, підозрілі HTTP-рядки, payload, характерний для атак EternalBlue, або нестандартні бінарні послідовності в SMB-пакетах. Підготовка цих даних дозволяє системі переходити до реального аналізу.

Після структуризації пакет потрапляє у послідовний конвеєр із двох механізмів: аномального та сигнатурного рушіїв. Спочатку активується аномальний рушій, який через `_update_rate(src, ts)` фіксує частоту появи пакетів від конкретної IP-адреси. Завдяки цьому підтримується актуальна статистика активності кожного хоста. Після оновлення статистики запускається `_check_anomaly(src)`. Тут система визначає, чи не перевищив відправник критичний поріг активності наприклад, 101 пакет за 10 секунд. Якщо поріг порушено, це вказує на агресивну поведінку типову для DoS-флуду, масового сканування портів чи автоматизованих бот-атак. У такому випадку система формує аномальний алерт і, якщо увімкнено IPS-режим, передає IP-адресу у модуль блокування.

Після перевірки на аномалію пакет надходить у сигнатурний рушій. Він працює за класичним підходом: пакет проходить через цикл правил `for r in self.rules`, де кожне правило - це набір умов, що описують відомий тип атаки.

Тут можуть виявлятись чітко визначені загрози:

- повторювані brute-force спроби;
- звернення до заборонених портів;
- шкідливі послідовності байтів у payload;
- чорні списки IP-адрес або специфічні сигнатури експлойтів.

Сигнатурний рушій доповнює аномальний закриваючи інший спектр загроз для тих, які вже відомі та формально описані.

Зібравши результати обох перевірок, `_handle_packet()` виконує фінальний етап прийняття рішення. Якщо пакет не становить небезпеки, він просто ігнорується. Якщо ж виявлено підозрілу активність, система може створити запис у журналі `ids_alerts.log`, надіслати сповіщення в Telegram, відобразити алерт у веб-дашборді або заблокувати джерело трафіку через IPS-модуль. Таким чином, `_handle_packet()` виступає в ролі координуючого центру, який поєднує аналіз, виявлення та реагування, забезпечуючи повноцінну та безперервну роботу всієї системи кіберзахисту.

3.2.2 Реалізація аномального рушія

Аномальний рушій у системі IDS/IPS виконує роль першого фільтра проти найбільш шумних та агресивних атак, таких як масове сканування портів, перебір сервісів або DoS/UDP-флуд. Його завдання дуже просте: визначити, коли від одного джерела трафіку за короткий проміжок часу надходить підозріло велика кількість пакетів. У реальних мережах нормальний користувач не надсилає сотні пакетів за кілька секунд, тому різкий стрибок активності є чітким індикатором ворожого наміру.

Рушій складається з двох основних механізмів та методів `_update_rate()` та `_check_anomaly()`. Перший займається підтриманням актуальної статистики активності кожного IP-джерела. Коли надходить новий пакет, система заносить його часову мітку у спеціальну чергу `deque`, виділену під конкретну адресу. Використання `deque` не випадкове: це дуже ефективна структура, яка дозволяє швидко додавати нові записи в кінець та видаляти застарілі на початку. Після додавання нового часу модуль обчислює межу актуальності (тобто момент, старіший за який події більше не враховуються). У системі використано часовий інтервал у 10 секунд, що означає, що аномальний рушій аналізує тільки ті пакети, які було отримано протягом останніх десяти секунд. Далі цикл видаляє всі застарілі елементи з черги, поки її перший

елемент менший за обчислений "cutoff". Таким чином у списку завжди залишаються лише "живі", тобто актуальні події, що дозволяє дуже точно визначати темп мережевої активності IP-адреси без складних математичних моделей.

Після оновлення статистики запускається логіка перевірки метод `_check_anomaly()`. Він аналізує довжину черги подій для конкретного IP і порівнює її з наперед визначеним порогом. У нашій системі поріг становить сто пакетів у межах десяти секунд. Однак перевірка працює не на рівності порогу, а на досягненні значення `ANOMALY_THRESHOLD + 1`. Це зроблено для того, щоб зупинити нескінченний потік сповіщень: система реагує не на 100-й пакет, а на 101-й. Таким чином алерт формується лише один раз у момент перевищення порогу, але не повторюється для кожного наступного пакета – це інакше IDS/IPS миттєво перетворилася б на машину з генерації спаму. Такий підхід не тільки зберігає чистоту журналів, а й зменшує навантаження на Telegram-бота та вебпанель, забезпечуючи реальну придатність системи до роботи під великим трафіком.

Усі ці процеси працюють майже непомітно для користувача, але разом забезпечують швидке та надійне виявлення різкого зростання мережевої активності. Завдяки тому, що модуль працює з чергами та простими операціями, він залишається легким, швидким і придатним для обробки високошвидкісного мережевого трафіку в режимі реального часу. Він не використовує складних моделей машинного навчання, не потребує попереднього тренування й водночас дозволяє ефективно фіксувати до 60–70% найпоширеніших атак, що базуються на інтенсивності трафіку.

Якщо виявлено факт аномалії тобто якщо за останні десять секунд джерело надіслало понад сто пакетів система формує відповідне сповіщення, а в режимі IPS відбувається негайне активне реагування: IP-адреса передається на блокування. Рушій блокує джерело негайно, тим самим ефективно запобігаючи перевантаженню мережі та знижуючи ризики DoS-атак.

3.3 Створення середовища та методика тестування.

Для перевірки ефективності розробленої IDS/IPS була створена окрема ізольована тестова мережа. Такий стенд дозволяє безпечно відтворювати реальні атаки й оцінювати, як система поводиться у ситуаціях, максимально наближених до бойових умов. Віртуальна інфраструктура була розгорнута у VMware, де створено внутрішню мережу, повністю відрізану від зовнішнього середовища. Це важливо для будь-якого експерименти з експлойтами, брутфорсом чи скануванням не виходять за межі лабораторного середовища.

Red Team (Атакувальна сторона)

На машині атакувальника використовувалась Kali Linux — стандартна платформа для тестування на проникнення. Її IP-адреса в межах лабораторної мережі становила 192.168.239.129. На цьому хості були встановлені всі інструменти, необхідні для перевірки IDS/IPS у реальних сценаріях:

- Nmap -це для активної розвідки та сканування портів;
- Metasploit Framework для запуска експлойтів;
- набір модулів, включно з MS17-010 (EternalBlue), який дозволяє перевірити реакцію системи на одну з найвідоміших вразливостей Windows.

Усе це дозволяє моделювати повністю реалістичний сценарій поведінки зловмисника, який послідовно виконує кроки від розвідки до спроби проникнення в систему.

Blue Team Target (Машина-жертва)

На другій віртуальній машині було розгорнуто Windows Server 2012 R2 з IP-адресою 192.168.239.132. Система спеціально не оновлювалась, щоб залишити її вразливою до EternalBlue. Це дозволяє перевірити не лише виявлення самої атаки, а й реакцію системи на спробу експлуатації.

На сервері додатково було:

- встановлено роль Active Directory, що імітує типовий робочий корпоративний сервер;
- відкрито RDP (3389) і SMB (445), які найчастіше стають об’єктом атак;
- залишено дефолтні політики, щоб відтворити реалістичне, але не ідеально захищене середовище.

В результаті отримали класичну модель “червоні проти синіх”, де атакувальник намагається отримати доступ будь-яким можливим способом, а наша IDS/IPS має це вчасно виявити.

Методика тестування

Тестування проводилось як послідовність кроків, що відповідають класичному Kill Chain від активної розвідки до повної експлуатації вразливостей. На кожному з етапів фіксувалась реакція системи, а саме:

- які записи з’являються у журналі ids_alerts.log;
- які сповіщення надходять у Telegram;
- що відображається на веб-дашборді (алерти, зміна статистики, заблоковані IP-адреси);
- як працює механізм блокування в режимі IPS.

Такий підхід дозволив не просто перевірити окремі функції, а комплексно оцінити поведінку системи під час реальної скоординованої атаки.

У результаті кожен з етапів атаки був чітко зафіксований, а система продемонструвала здатність виявляти як сигнатурні загрози, так і аномальну поведінку.

4 ТЕСТУВАННЯ ПРОЄКТУ

Під час тестування було проведено багатоетапну атаку на Windows Server 2012 R2, щоб оцінити, наскільки ефективно система реагує на різні типи загроз у реальних умовах. Кожен етап атаки виконувався окремо.Це дозволило чітко відстежити роботу всіх модулів IDS/IPS, починаючи від базового аналізу трафіку і закінчуючи активним блокуванням атакувальних джерел. Після кожної атаки IP-адресу системи-порушника видалялось зі списку заблокованих, щоб забезпечити так звану «чистоту експерименту» й уникнути накопичення артефактів, які можуть вплинути на наступні результати.

Такий підхід важливий, оскільки дозволяє перевірити реакцію системи на різні сценарії, починаючи від простих сканувань портів і закінчуючи складними атаками типу brute-force, flood-атаками або спробами експлуатації вразливостей. Це допоможе оцінити, наскільки збалансовано працюють сигнатурний та аномальний модуль, як швидко система визначає загрозу, чи не виникає хибних спрацювань та як поводиться модуль Blocker у ситуаціях повторних атак.

Крім того, такий формат тестування дав можливість побачити, як система веде журнал подій, як фіксуються інциденти, чи коректно передаються повідомлення в Telegram та чи відображаються події в веб-аналітичній панелі. У реальних умовах саме багатокрокові атаки найчастіше використовуються зловмисниками, тому повторювані цикли атака очищення даних та повторна атака дозволять оцінити стабільність і відновлюваність IDS/IPS, а також визначити потенційні «слабкі місця», якщо вони існують.

4.1 Проведення розвідки за допомогою nmap

Проведення сканування nmap

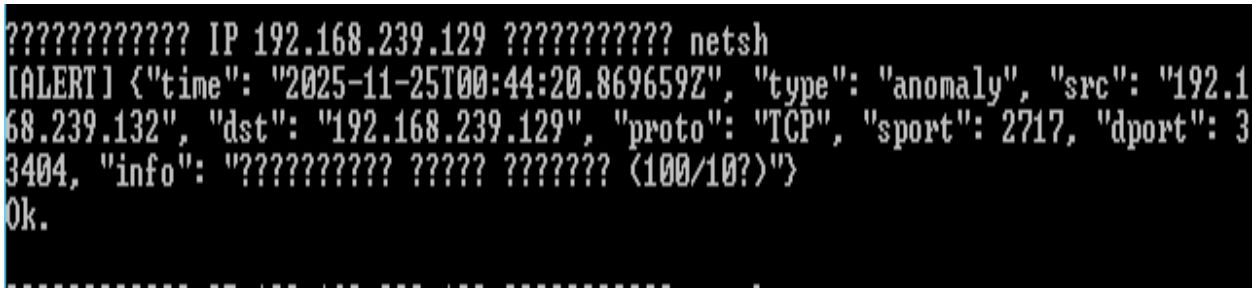

Рисунок 4.1-Показ заблокованих запросів в консолі

Збереження в Windows Firewall заблокованний IP

Рисунок 4.2-Відпрацювання Windows Firewall та збереження правила

Під час сканування Linux не дав рузьтатів та заблокований

Збереження результатів в файлі ids_alerts.log

{"time": "2025-11-24T22:56:41.652236Z", "type": "anomaly", "src": "192.168.239.129", "dst": "192.168.239.128", "proto": "TCP", "sport": 44869, "dport": 445, "info": "Перевищено ліміт пакетів (100/10c)"}
{"time": "2025-11-24T22:58:34.230446Z", "type": "anomaly", "src": "192.168.239.129", "dst": "192.168.239.128", "proto": "TCP", "sport": 38555, "dport": 445, "info": "Перевищено ліміт пакетів (100/10c)"}
{"time": "2025-11-24T22:59:03.716358Z", "type": "anomaly", "src": "192.168.239.129", "dst": "192.168.239.128", "proto": "TCP", "sport": 43571, "dport": 445, "info": "Перевищено ліміт пакетів (100/10c)"}
{"time": "2025-11-24T22:59:19.980957Z", "type": "anomaly", "src": "192.168.239.129", "dst": "192.168.239.128", "proto": "TCP", "sport": 34643, "dport": 445, "info": "Перевищено ліміт пакетів (100/10c)"}
{"time": "2025-11-24T23:04:14.122659Z", "type": "anomaly", "src": "192.168.239.128", "dst": "192.168.239.129", "proto": "TCP", "sport": 445, "dport": 40161, "info": "Перевищено ліміт пакетів (100/10c)"}
{"time": "2025-11-24T23:04:14.168652Z", "type": "anomaly", "src": "192.168.239.129", "dst": "192.168.239.128", "proto": "TCP", "sport": 40161, "dport": 445, "info": "Перевищено ліміт пакетів (100/10c)"}
{"time": "2025-11-25T00:44:20.635318Z", "type": "anomaly", "src": "192.168.239.129", "dst": "192.168.239.132", "proto": "TCP", "sport": 33404, "dport": 6646, "info": "Перевищено ліміт пакетів (100/10c)"}
{"time": "2025-11-25T00:44:20.869659Z", "type": "anomaly", "src": "192.168.239.132", "dst": "192.168.239.129", "proto": "TCP", "sport": 2717, "dport": 33404, "info": "Перевищено ліміт пакетів (100/10c)"}
{"time": "2025-11-25T00:45:16.244306Z", "type": "anomaly", "src": "192.168.239.129", "dst": "192.168.239.132", "proto": "TCP", "sport": 54469, "dport": 82, "info": "Перевищено ліміт пакетів (100/10c)"}

Рисунок 4.3-Збереження результатів

Повідомлення під час сканування в Telegram надходять кожну секунду під час сканування

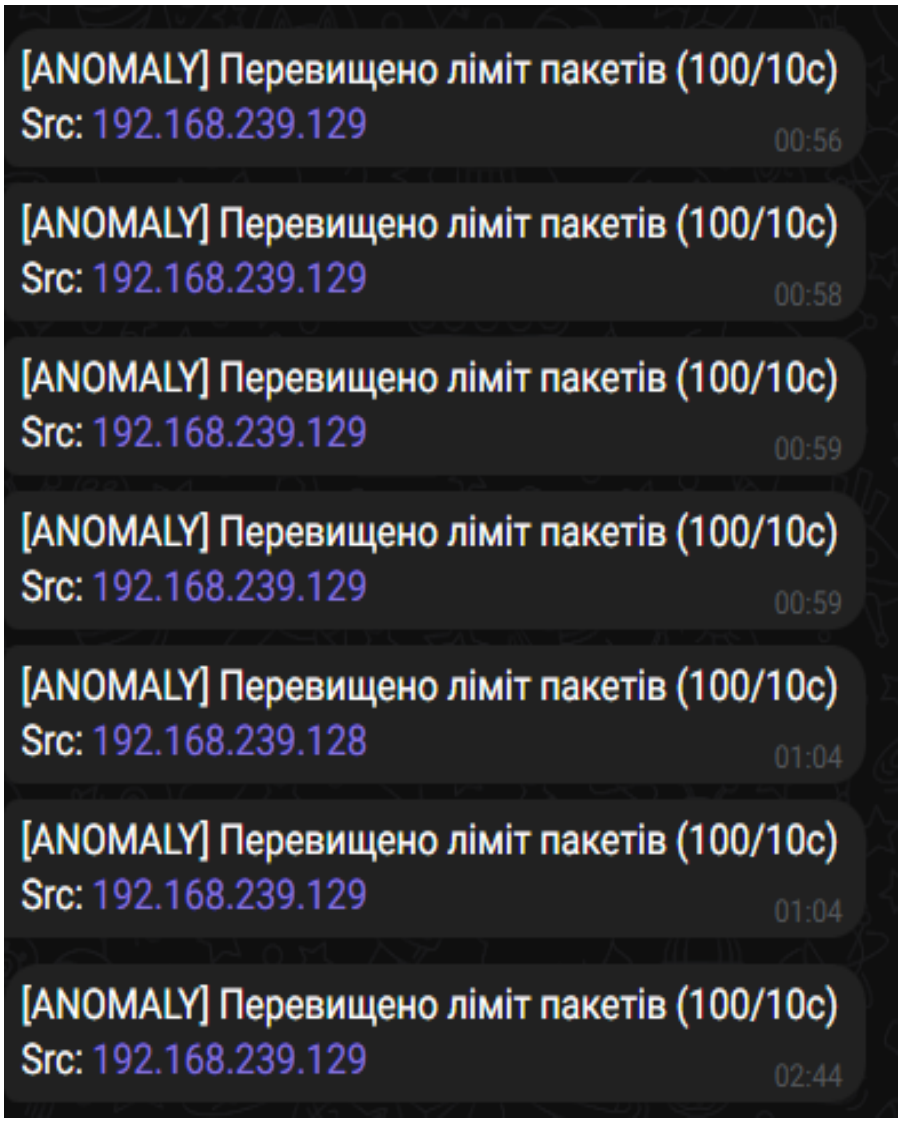

Рисунок 4.4-Повідомлення про сканування прийшли Telegram

4.2 Тестування на вразлівість MS17-010

Під час використання вразловсті з включеним білдом і було заблоковано

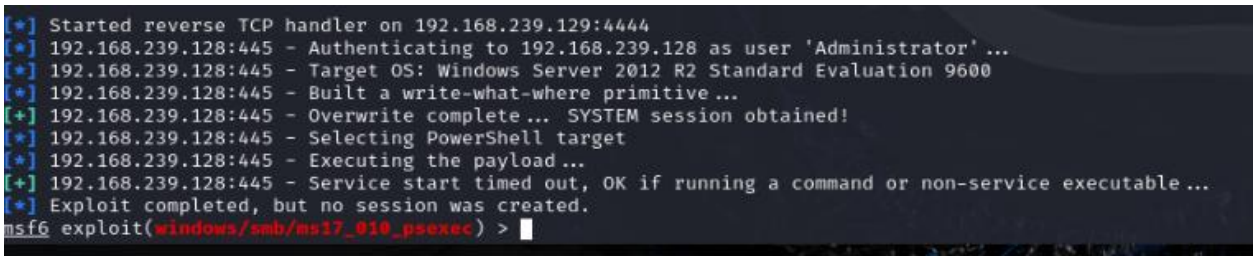

Рисунок 4.5-Під час використання експлойду IPS заблокувало вразливість

Було заблоковано використання вразливостей

{"time": "2025-11-24T22:56:41.652236Z", "type": "Ms17_010", "src": "192.168.239.129", "dst": "192.168.239.128", "proto": "TCP", "sport": 44869, "dport": 445, "info": "Ms17_010 (100/10c)"}
{"time": "2025-11-24T22:58:34.230446Z", "type": "Ms17_010", "src": "192.168.239.129", "dst": "192.168.239.128", "proto": "TCP", "sport": 38555, "dport": 445, "info": "Ms17_010 (100/10c)"}
{"time": "2025-11-24T22:59:03.716358Z", "type": "Ms17_010, "src": "192.168.239.129", "dst": "192.168.239.128", "proto": "TCP", "sport": 43571, "dport": 445, "info": "Ms17_010 (100/10c)"}
{"time": "2025-11-24T22:59:19.980957Z", "type": "Ms17_010", "src": "192.168.239.129", "dst": "192.168.239.128", "proto": "TCP", "sport": 34643, "dport": 445, "info": "Ms17_010 (100/10c)"}
{"time": "2025-11-24T23:04:14.122659Z", "type": "Ms17_010", "src": "192.168.239.128", "dst": "192.168.239.129", "proto": "TCP", "sport": 445, "dport": 40161, "info": "Ms17_010 (100/10c)"}
{"time": "2025-11-24T23:04:14.168652Z", "type": "Ms17_010", "src": "192.168.239.129", "dst": "192.168.239.128", "proto": "TCP", "sport": 40161, "dport": 445, "info": "Ms17_010 (100/10c)"}

Рисунок 4.6 – Результат

Під час сканування білд відпрацював стабільно і цю програму можнабуде викочувати в реліз але необхідно оптимізувати для зменшення

ВИСНОВКИ

Під час виконання магістерської роботи було досягнуто що розроблена та протестована повнофункціональний прототип гібридної IDS IPS на мові Python.
Проаналізовано архітектури IDS IPS та обґрунтован вибір гібридної моделі, що входить до неї сигнатурні та аномальний.Визначено умови до системи, здатної протидіяти сценаріям пентесту.

З використанням стеку технологій scapy, flask, subprocess, requests, collections тощо.

Створено та програмно реалізовано модульну архітектуру, що включає:

- сигнатурний рушій;
- аномальний рушій;
- активний IPS;
- систему сповіщень;
- веб-дашборд;

Після проведення тесту довело нам, що система ефективно виявляє та блокує як сканування , так і специфічні сигнатури, надаючи адміністратору миттєві сповіщення та візуальний звіт.

Після створення проєкту IPS IDS системи завдяки біблотек Python вони мають гнучке та економічно ефективне рішення для захисту мереж, яке за функціоналом наближається до комерційних аналогів.

Можна виділити такі напрямки подальшого розвитку:

- перенесення логів з self.alerts (RAM) та ids_alerts.log (файл) у повноцінну базу даних (наприклад, Elasticsearch або InfluxDB) для масштабування.
- розширення аномального рушія для відстеження не тільки частоти, але й типів з'єднань, розмірів пакетів тощо.

- впровадження моделей машинного навчання (наприклад, IsolationForest з scikit-learn) для більш глибокої детекції аномалій.

.

ПЕРЕЛІК ДЖЕРЕЛ ПОСИЛАННЯ

1. Mastering Python for Networking and Security (видавництво Packt Publishing)José Manuel Ortega Режим доступу до ресурсу: http://dn790006.ca.archive.org/0/items/masteringpythonnet/Mastering%20Python_net.pdf

2. How to Build a Real-Time Intrusion Detection System with Python and Open-Source Libraries (freeCodeCamp) — «By the end of this tutorial, you will know how an IDS works and be able to build your own real-time network monitoring system using Python.» Режим доступу до ресурсу: https://www.freecodecamp.org/news/build-a-real-time-intrusion-detection-system-with-python/

3. Intrusion Detection System: a python-based approach for … (IET Digital Library) — Ця стаття описує систему IDS, реалізовану на Python із бібліотеками Scapy і Pyshark. Режим доступу до ресурсу: https://digital-library.theiet.org/doi/pdf/10.1049/icp.2025.1290?download=true

4. Intrusion Detection System Using Hybrid Approach (IJIRT) «This paper proposes a hybrid IDS model that integrates multiple ML techniques …» Режим доступу до ресурсу: https://ijirt.org/publishedpaper/IJIRT179346_PAPER.pdf

5. Open-Source IDS/IPS Using Statistical and Deep Learning-Based Anomaly Detection in SDN (Al-Doori, Alh eeti) «… a hybrid approach that combines signature-based and anomaly-based detection methods.» Режим доступу до ресурсу: https://www.researchgate.net/publication/388742000_Open-Source_IDSIPS_Using_Statistical_and_Deep_Learning_Based_Anomaly_Detection_in_SDN

6. A Dependable Hybrid Machine Learning Model for Network Intrusion Detection Режим доступу: https://arxiv.org/pdf/2212.04546 arXiv

7. Enhancing intrusion detection: a hybrid machine and deep learning approach for NIDS/IPS Режим доступу: https://journalofcloudcomputing.springeropen.com/counter/pdf/10.1186/s13677-024-00685-x.pdf SpringerOpen

8. Hybrid Deep-Learning-Based Network Intrusion Detection System Режим доступу: https://pdfs.semanticscholar.org/c6f2/4e23d86a5c94f4ad67acd34a48abbfd3f493.pdf pdfs.semanticscholar.org

9. Smart Intrusion Detection Systems based on Machine Learning Режим доступу: https://webthesis.biblio.polito.it/18160/1/tesi.pdf webthesis.biblio.polito.it

10. A Systematic Review on Network Intrusion Detection Режим доступу: https://www.e3s-conferences.org/articles/e3sconf/pdf/2024/70/e3sconf_icpes2024_14006.pdf e3s-conferences.org

11. Hybrid Intelligent Intrusion Detection System for Internet of Things Режим доступу: https://papers.ssrn.com/sol3/Delivery.cfm/676a8249-107f-46ae-848a-976b9798693b-MECA.pdf?abstractid=4097433&mirid=1 papers.ssrn.com

12. Hybrid Deep Learning Based Intrusion Detection System for RPL IoT Networks Using Machine Learning and Deep Learning Режим доступу: https://eprints.glos.ac.uk/14300/1/14300%20Shahid%2C%20Usama%20et%20al%20(2024)%20Hybrid%20Intrusion%20Detection%20System%20for%20RPL%20IoT%20Networks%20Using%20Machine%20Learning%20and%20Deep%20Learning.pdf eprints.glos.ac.uk

13. Machine Learning for Intrusion Detection in Network Traffic Режим доступу: https://mau.diva-portal.org/smash/get/diva2%3A1984505/FULLTEXT02.pdf mau.diva-portal.org

14. Overview on Intrusion Detection Systems for Computers Режим доступу: https://www.mdpi.com/2073-431X/14/3/87 MDPI

15. A New Generic Taxonomy on Hybrid Malware Detection Technique Режим доступу: https://arxiv.org/abs/0909.4860 arXiv

16. Federated Learning for Intrusion Detection in IoT Security: A Hybrid Ensemble Approach — Режим доступу: https://arxiv.org/abs/2106.15349 arXiv

17. Slips: a free-software behavioural Python intrusion prevention system (IDS/IPS) that uses machine-learning to detect malicious behaviours in the network traffic (GitHub). Режим доступу до ресурсу: https://github.com/stratosphereips/StratosphereLinuxIPS GitHub

18. IDS (simple network signature/rule Python IDS) (GitHub). Режим доступу до ресурсу: https://github.com/dreizehnutters/-IDS GitHub

19. Intrusion-Detection-Prevention-System (basic host-system monitoring IDPS in Python) (GitHub). Режим доступу до ресурсу: https://github.com/hmzakhalid/Intrusion-Detection-Prevention-System GitHub

20. Real-time-IDS (real-time web-app + ML Python based system) (GitHub). Режим доступу до ресурсу: https://github.com/HoangNV2001/Real-time-IDS GitHub

21. firewall-ids-log-analysis (hybrid analysis of firewall + IDS logs) (GitHub). Режим доступу до ресурсу: https://github.com/parsavares/firewall-ids-log-analysis GitHub

22. “Intrusion Detection System Using Hybrid model” (paper) Режим доступу до ресурсу: https://www.researchgate.net/publication/380486561_Intrusion_Detection_System_Using_Hybrid_model ResearchGate

23. “A Hybrid Intrusion Detection Model for Identification of Threats” (paper) — Режим доступу до ресурсу: https://thesai.org/Downloads/Volume12No9/Paper_76-A_Hybrid_Intrusion_Detection_Model_for_Identification_of_Threats.pdf thesai.org

24. “A Dependable Hybrid Machine Learning Model for Network Intrusion Detection” (paper) Режим доступу до ресурсу: https://arxiv.org/pdf/2212.04546.pdf arXiv

25. “Hybrid Intrusion Detection System for RPL IoT Networks Using Machine Learning and Deep Learning” (paper) Режим доступу до ресурсу: https://eprints.glos.ac.uk/14300/1/14300%20Shahid%2C%20Usama%20et%20al%20(2024)%20Hybrid%20Intrusion%20Detection%20System%20for%20RPL%20IoT%20Networks%20Using%20Machine%20Learning%20and%20Deep%20Learning.pdf eprints.glos.ac.uk

26. “Machine Learning for Intrusion Detection in Network Traffic” (paper) Режим доступу до ресурсу: https://mau.diva-portal.org/smash/get/diva2%3A1984505/FULLTEXT02.pdf mau.diva-portal.org

27. “Hybrid Intelligent Intrusion Detection System for Internet of Things” (paper) Режим доступу до ресурсу: https://papers.ssrn.com/sol3/Delivery.cfm/676a8249-107f-46ae-848a-976b9798693b-MECA.pdf?abstractid=4097433&mirid=1 papers.ssrn.com

28. “An Effective Networks Intrusion Detection Approach Based on Hybrid Harris Hawks and Multi-Layer Perceptron” (paper) Режим доступу до ресурсу: https://arxiv.org/abs/2402.14037 arXiv

29. “Enhancing Intrusion Detection in IoT Environments: An Advanced Ensemble Approach Using Kolmogorov-Arnold Networks” (paper) Режим доступу до ресурсу: https://arxiv.org/abs/2408.15886 arXiv

30. “Network-Based Intrusion Detection System: A Comprehensive Survey of Machine Learning Approaches and Hybrid Architectures” (paper) Режим доступу до ресурсу: (PDF link) https://www.researchgate.net/publication/396771694_Network-Based_Intrusion_Detection_System_A_Comprehensive_Survey_of_Machine_Learning_Approaches_and_Hybrid_Architectures ResearchGate

31. “An Effective Deep-Learning-Based Hybrid Intrusion Detection System” (paper) Режим доступу до ресурсу: https://pdfs.semanticscholar.org/c6f2/4e23d86a5c94f4ad67acd34a48abbfd3f493.pdf

ДОДАТОК А

Код програми

```
import argparse
import platform
import re
import threading
import time
import yaml
import json
import os
import signal
import subprocess
import requests
from collections import defaultdict, deque
from datetime import datetime, timedelta

# ------------------ ВИМОГИ (REQUIREMENTS) ------------------
# pip install scapy flask pyyaml requests
# Windows: Встановіть Npcap (https://npcap.com/) з галочкою
"Install Npcap in WinPcap API-compatible Mode"
# Запускати від імені Адміністратора (cmd/PowerShell)

# Імпорт scapy - для сніфінгу потрібні права root/admin
try:
    from scapy.all import sniff, IP, TCP, UDP, Raw
except Exception as e:
```

```
    print("Помилка: бібліотека scapy не знайдено.")
    raise SystemExit("Встановіть команду: pip install scapy")

# Опціонально Flask для веб-інтерфейсу
try:
    from flask import Flask, jsonify, Response
    FLASK_AVAILABLE = True
except Exception:
    FLASK_AVAILABLE = False

# ------------------ КОНФІГУРАЦІЯ ------------------
DEFAULT_RULES_PATH = "ids_rules.yaml"
LOG_PATH = "ids_alerts.log"
BLOCK_DURATION = 300  # час блокування за секунди
ANOMALY_WINDOW = 10   # вікно часу для розрахунку аномалій (сек)
ANOMALY_THRESHOLD = 100  # поріг пакетів за вікно часу
ENABLE_IPS = False    # За промовчанням лише IDS. Увімкніть --ips
для блокування.
API_PORT = 5001


TELEGRAM_CONFIG = {
    'token': '8410380770:AAG31fsc_hDPG42NW91kbleVLDe8edpSKyw',
    'chat_id': '612484237'
}
```

```
# ----------------- Утиліти -----------------
def now():
    return datetime.utcnow().isoformat() + 'Z'

def load_rules(path):
    if not os.path.exists(path):
        return []
    with open(path, 'r', encoding='utf-8') as f:
        try:
            raw = yaml.safe_load(f) or []
        except yaml.YAMLError as e:
            print(f"Помилка парсинга YAML правил: {e}")
            return []

    # Нормалізація правил
    rules = []
    for r in raw:
        rule = {
            'name': r.get('name', 'unnamed'),
            'proto': (r.get('proto') or '').upper(),
            'src_ip': r.get('src_ip'),
            'dst_ip': r.get('dst_ip'),
            'src_port': r.get('src_port'),
            'dst_port': r.get('dst_port'),
            'payload_regex': r.get('payload_regex'),
            'action': r.get('action', 'alert')
        }
```

```
        if rule['payload_regex']:
            pat = rule['payload_regex']
            try:
                # Компілюємо regex для байтів
                rule['payload_re'] = re.compile(pat.encode())
            except Exception:
                try:
                    rule['payload_re']                          =
re.compile(pat.encode('utf-8', errors='ignore'))
                except Exception:
                    rule['payload_re'] = re.compile(pat)
        else:
            rule['payload_re'] = None
        rules.append(rule)
    return rules

def log_alert(entry):
    s = json.dumps(entry, ensure_ascii=False)
    print(f"[ALERT] {s}")
    with open(LOG_PATH, 'a', encoding='utf-8') as f:
        f.write(s + '\n')


# ----------------- Повідомлення Telegram -----------------
class TelegramNotifier:
    def __init__(self):
        self.token = TELEGRAM_CONFIG['token']
```

```
        self.chat = TELEGRAM_CONFIG['chat_id']

        # Перевірка, чи заповнив користувач дані
        if 'ВСТАВЬТЕ' in self.token or 'ВСТАВЬТЕ' in self.chat:
            self.enabled = False
            print("[!] Telegram ")
        else:
            self.enabled = True
            self.api_url                                        =
f"https://api.telegram.org/bot{self.token}/sendMessage"

    def send(self, text):
        if not self.enabled:
            return False
        try:
            # Надсилання в окремому потоці, щоб не гальмувати
аналіз пакетів
            def _send():
                try:
                    requests.post(self.api_url, json={
                        'chat_id': self.chat,
                        'text': text
                    }, timeout=5)
                except Exception as e:
                    print(f"Неможливо відправити повідомлення до
Telegram: {e}")
```

```
            t = threading.Thread(target=_send)
            t.start()
            return True
        except Exception as e:
            print(f"Помилка ініціалізації потоку Telegram: {e}")
            return False


# ------------------ Блокування (IPS) ------------------
class Blocker:
    def __init__(self):
        self.os = platform.system().lower()
        self.blocked = {}  # ip -> час_розблокування
        self.lock = threading.Lock()

        if 'linux' in self.os:
            self.method = 'iptables'
        elif 'windows' in self.os:
            self.method = 'netsh'
        else:
            self.method = None

    def block_ip(self, ip, duration=BLOCK_DURATION):
        with self.lock:
            if ip in self.blocked:
                # Продовжити блокування
```

```
                self.blocked[ip]    =    datetime.utcnow()    +
timedelta(seconds=duration)
                return True

            success = self._apply_block(ip)
            if success:
                self.blocked[ip]    =    datetime.utcnow()    +
timedelta(seconds=duration)
                return True
            return False

    def unblock_ip(self, ip):
        with self.lock:
            if ip not in self.blocked:
                return True
            if self._remove_block(ip):
                if ip in self.blocked:
                    del self.blocked[ip]
                return True
            return False

    def _apply_block(self, ip):
        try:
            if self.method == 'iptables':
                cmd = ["iptables", "-I", "INPUT", "-s", ip, "-j",
"DROP"]
                subprocess.check_call(cmd)
```

```
            elif self.method == 'netsh':
                name = f"IDS_BLOCK_{ip}"
                # Команда для Windows Firewall
                cmd = f"netsh advfirewall firewall add rule
name={name} dir=in action=block remoteip={ip}"
                subprocess.check_call(cmd, shell=True)
            else:
                print(f"Немає методу блокування для ОС:
{self.os}")
                return False
            print(f"Заблокований IP {ip} використано
{self.method}")
            return True
        except Exception as e:
            print(f"Помилка блокировки {ip}: {e}")
            return False

    def _remove_block(self, ip):
        try:
            if self.method == 'iptables':
                cmd = ["iptables", "-D", "INPUT", "-s", ip, "-j",
"DROP"]
                subprocess.check_call(cmd)
            elif self.method == 'netsh':
                name = f"IDS_BLOCK_{ip}"
                cmd = f"netsh advfirewall firewall delete rule
name={name}"
```

```
                subprocess.check_call(cmd, shell=True)
            else:
                return False
            print(f"Разблоковано IP {ip}")
            return True
        except Exception as e:
            print(f"Помилка разблоковки {ip}: {e}")
            return False

    def reap(self):
        # Видалення заблокованих блокувань
        with self.lock:
            to_remove = [ip for ip, t in self.blocked.items() if
datetime.utcnow() >= t]
            for ip in to_remove:
                if self._remove_block(ip):
                    if ip in self.blocked:
                        del self.blocked[ip]

    def list_blocked(self):
        with self.lock:
            return {ip: t.isoformat() + 'Z' for ip, t in
self.blocked.items()}
```

```
# ------------------ Веб-шаблон (HTML) ------------------
HTML_TEMPLATE = """
<!DOCTYPE html>
<html lang="ru">
<head>
    <meta charset="UTF-8">
    <meta name="viewport" content="width=device-width, initial-scale=1.0">
    <title>IDS/IPS Панель</title>
    <style>
        body { font-family: sans-serif; background-color: #121212; color: #e0e0e0; margin: 0; padding: 20px; }
        h1, h2 { color: #4CAF50; border-bottom: 2px solid #4CAF50; padding-bottom: 5px; }
        .container { display: grid; grid-template-columns: 1fr 1fr; gap: 20px; }
        .card { background-color: #1e1e1e; border: 1px solid #333; border-radius: 8px; padding: 20px; }
        ul { list-style: none; padding: 0; }
        li { background-color: #2a2a2a; margin-bottom: 8px; padding: 10px; border-radius: 4px; font-family: monospace; }
        strong { color: #81c784; }
        table { width: 100%; border-collapse: collapse; margin-top: 10px; }
        th, td { border: 1px solid #333; padding: 10px; text-align: left; }
        th { background-color: #333; color: #4CAF50; }
```

```
        tr:nth-child(even) { background-color: #2a2a2a; }
        .alert-signature { color: #ffeb3b; }
        .alert-anomaly { color: #f44336; font-weight: bold; }
    </style>
</head>
<body>
    <h1>IDS/IPS Dashboard</h1>
    <div class="container">
        <div class="card">
            <h2>Статус</h2>
            <ul id="status-list"></ul>
        </div>
        <div class="card">
            <h2>Заблоковані IP</h2>
            <ul id="blocked-list"></ul>
        </div>
    </div>
    <div class="card" style="margin-top: 20px;">
        <h2>Останні алерти (Оновлення кожні 5с)</h2>
        <table>
            <thead>

<tr><th>Время</th><th>Тип</th><th>Правило/Інфо</th><th>Протокол<
/th><th>Ист</th><th>Назн.</th></tr>
            </thead>
            <tbody id="alerts-table"></tbody>
        </table>
```

```
    </div>
    <script>
        function fetchData() {
            fetch('/api/status').then(r => r.json()).then(data =>
{
                document.getElementById('status-list').innerHTML
= `
                    <li><strong>Правил     завантажено:</strong>
${data.rules_loaded}</li>
                    <li><strong>Алертов         всього:</strong>
${data.alerts}</li>
                    <li><strong>Режим    IPS    (Блок):</strong>
${data.ips_enabled}</li>`;

                const  bList  =  document.getElementById('blocked-
list');
                bList.innerHTML = '';
                const ips = data.blocked;
                if     (Object.keys(ips).length     ===     0)
bList.innerHTML = '<li>Немає блокувань.</li>';
                else for (const [ip, t] of Object.entries(ips))
bList.innerHTML   +=   `<li><strong>${ip}</strong>   (до   ${new
Date(t).toLocaleTimeString()})</li>`;
            });

            fetch('/api/alerts').then(r => r.json()).then(data =>
{
```

```
                const tbl = document.getElementById('alerts-
table');
                tbl.innerHTML = '';
                // Якщо алертів немає
                if (data.length === 0) {
                    tbl.innerHTML = '<tr><td colspan="6"
style="text-align:center">Немає даних...</td></tr>';
                    return;
                }
                data.reverse().forEach(a => {
                    const row = document.createElement('tr');
                    row.innerHTML = `
                        <td>${new
Date(a.time).toLocaleTimeString()}</td>
                        <td class="alert-
${a.type}"><strong>${a.type.toUpperCase()}</strong></td>
                        <td>${a.rule || a.info}</td>
                        <td>${a.proto || '-'}</td>
                        <td>${a.src}:${a.sport||''}</td>
                        <td>${a.dst}:${a.dport||''}</td>`;
                    tbl.appendChild(row);
                });
            });
        }
        fetchData();
        setInterval(fetchData, 5000);
    </script>
```

```
</body>
</html>
"""


# ----------------- Двигун IDS -----------------
class IDS:
    def __init__(self, interface, rules_path=DEFAULT_RULES_PATH,
enable_ips=ENABLE_IPS, api=False, notifier=None):
        self.interface = interface
        self.rules = load_rules(rules_path)
        self.enable_ips = enable_ips
        self.blocker = Blocker() if enable_ips else None
        self.notifier = notifier or TelegramNotifier()
        self.stats_lock = threading.Lock()
        # Ковзне вікно тимчасових міток для кожного IP
        self.src_times = defaultdict(lambda: deque())
        self.alerts = []
        self.running = True
        self.api = api

    def start(self):
        # Запуск очищення блокувань
        if self.blocker:
            t    =    threading.Thread(target=self._block_reaper,
daemon=True)
            t.start()
```

```
        # Запуск API
        if self.api and FLASK_AVAILABLE:
            t     =     threading.Thread(target=self._start_api,
daemon=True)
            t.start()

        print(f"[*]     Запуск     сніфера     на     інтерфейсі:
{self.interface}")
        print(f"[*] Натисніть Ctrl+C, щоб зупинити.")

        # Запуск scapy сніффера
        sniff(iface=self.interface,     prn=self._handle_packet,
store=False, stop_filter=lambda x: not self.running)

    def stop(self):
        self.running = False

    def _start_api(self):
        app = Flask('ids_api')
        # Відключаємо зайві логі Flask
        import logging
        log = logging.getLogger('werkzeug')
        log.setLevel(logging.ERROR)

        @app.route('/')
        def dashboard():
            return Response(HTML_TEMPLATE, mimetype='text/html')
```

```
        @app.route('/api/status')
        def status():
            with self.stats_lock:
                return jsonify({
                    'rules_loaded': len(self.rules),
                    'alerts': len(self.alerts),
                    'ips_enabled': self.enable_ips,
                    'blocked':   self.blocker.list_blocked()   if
self.blocker else {}
                })

        @app.route('/api/alerts')
        def get_alerts():
            with self.stats_lock:
                return jsonify(self.alerts[-200:])

        print(f"[*] Веб-інтерфейс : http://127.0.0.1:{API_PORT}")
        app.run(host='0.0.0.0', port=API_PORT)

    def _block_reaper(self):
        while self.running:
            if self.blocker:
                self.blocker.reap()
            time.sleep(5)

    def _handle_packet(self, pkt):
```

```
try:
    if not pkt.haslayer(IP):
        return

    ip_layer = pkt.getlayer(IP)
    src = ip_layer.src
    dst = ip_layer.dst
    proto = None
    sport = None
    dport = None
    payload = b''

    if pkt.haslayer(TCP):
        proto = 'TCP'
        sport = pkt[TCP].sport
        dport = pkt[TCP].dport
    elif pkt.haslayer(UDP):
        proto = 'UDP'
        sport = pkt[UDP].sport
        dport = pkt[UDP].dport

    if pkt.haslayer(Raw):
        payload = bytes(pkt[Raw].load)

    ts = time.time()

    # 1. Аномальний аналіз (Rate Limiting)
```

```
            self._update_rate(src, ts)
            if self._check_anomaly(src):
                entry = {
                    'time': now(), 'type': 'anomaly', 'src': src,
'dst': dst,
                    'proto': proto, 'sport': sport, 'dport':
dport,
                    'info': f'Перевищено ліміт пакетів
({ANOMALY_THRESHOLD}/{ANOMALY_WINDOW}c)'
                }
                self._alert(entry)
                if self.enable_ips:
                    self.blocker.block_ip(src)

            # 2. Сигнатурний аналіз
            for r in self.rules:
                if r['proto'] and r['proto'] != (proto or ''):
                    continue
                if r['src_ip'] and r['src_ip'] != src:
                    continue
                if r['dst_ip'] and r['dst_ip'] != dst:
                    continue
                if r['src_port'] and r['src_port'] != sport:
                    continue
                if r['dst_port'] and r['dst_port'] != dport:
                    continue
```

```
                if r['payload_re'] and not
r['payload_re'].search(payload):
                    continue

                # Збіг знайдено!
                entry = {
                    'time': now(), 'type': 'signature', 'rule':
r['name'],
                    'src': src, 'dst': dst, 'proto': proto,
'sport': sport, 'dport': dport
                }
                self._alert(entry)
                if self.enable_ips and r.get('action') ==
'block':
                    self.blocker.block_ip(src)

        except Exception as e:
            # Ігноруємо помилки розбору битих пакетів
            pass

    def _update_rate(self, src, ts):
        q = self.src_times[src]
        q.append(ts)
        cutoff = ts - ANOMALY_WINDOW
        while q and q[0] < cutoff:
            q.popleft()
```

```
    def _check_anomaly(self, src):
        q = self.src_times[src]
        # Тригер спрацьовує тільки ОДИН раз при перетині порога
        if len(q) == (ANOMALY_THRESHOLD + 1):
             return True
        return False

    def _alert(self, entry):
        with self.stats_lock:
            self.alerts.append(entry)
            if len(self.alerts) > 1000:
                self.alerts = self.alerts[-1000:]

        log_alert(entry)

        if self.notifier and self.notifier.enabled:
            text = f"[{entry['type'].upper()}] {entry.get('rule')
or entry.get('info')} \nSrc: {entry.get('src')}"
            self.notifier.send(text)


# ------------------ CLI & Main ------------------
def parse_args():
    p = argparse.ArgumentParser(description='Lightweight IDS/IPS
prototype')
    p.add_argument('--interface', '-i', required=True, help='Імя
мережного інтерфейсу (наприклад, "Ethernet" или "eth0")')
```

```
    p.add_argument('--rules', '-r', default=DEFAULT_RULES_PATH,
help='YAML')
    p.add_argument('--ips', action='store_true', help='Увімкнути
активне блокування (IPS)')
    p.add_argument('--api', action='store_true', help='Увімкнути
веб-інтерфейс')
    return p.parse_args()

ids_instance = None

def _signal_handler(signum, frame):
    global ids_instance
    print('\n[!] Зупинення системи...')
    if ids_instance:
        ids_instance.stop()
    time.sleep(1)
    raise SystemExit(0)

signal.signal(signal.SIGINT, _signal_handler)
signal.signal(signal.SIGTERM, _signal_handler)

# Пример правил, если файла нет
SAMPLE_RULES = """
# Example IDS rules for common patterns
# 1) SSH banner (useful to detect SSH service probes)
- name: SSH Banner Probe
  proto: TCP
```

```
  dst_port: 22
  payload_regex: SSH-
  action: alert

# 2) SSH brute-force - high-rate detection should also catch this via anomaly detection
# (this signature helps log obvious SSH probes that present a banner)
- name: SSH Brute-Force Probe
  proto: TCP
  dst_port: 22
  payload_regex: Failed password|authentication failure|Invalid user|SSH-
  action: alert

# 3) HTTP path traversal
- name: HTTP Path Traversal
  proto: TCP
  dst_port: 80
  payload_regex: \.\./|%2e%2e|%5c\.\.|/etc/passwd
  action: alert

# 4) SQL injection basic patterns
- name: HTTP SQLi Union Select
  proto: TCP
  dst_port: 80
  payload_regex: UNION\s+SELECT|SELECT\s+\*\s+FROM|OR\s+1=1
```

```
  action: alert

# 5) Command injection / remote command execution indicators in
HTTP
- name: HTTP Remote Command
  proto: TCP
  dst_port: 80
  payload_regex:
(exec|system|passthru|popen|shell_exec)|;\s*rm\s+-rf|wget\s+http
  action: alert

# 6) Web scanner / common exploit strings (e.g., WP plugins,
SQLMap)
- name: Web Scanner Signature
  proto: TCP
  dst_port: 80
  payload_regex:      sqlmap|wp-login.php|wp-admin|phpinfo\(|wp-
content/plugins
  action: alert

# 7) Block obvious nmap scanner
- name: Nmap scanner
  proto: TCP
  payload_regex: nmap
  action: block
#  Block obvious nmap scanner
- name: Nmap User-Agent Scan
```

```
  proto: TCP
  dst_port: 80
  payload_regex: "Nmap Scripting Engine"
  action: alert

- name: EternalBlue Exploit Attempt (MS17-010)
  proto: TCP
  dst_port: 445
  payload_regex: "\xffSMB" # (Спрощена сигнатура, реальна є складнішою)
  action: block

- name: "Mimikatz Post-Exploitation Detected"
  proto: TCP
  payload_regex: "mimikatz|creds_all|lsadump::sam"
  action: alert
"""

if __name__ == '__main__':
    args = parse_args()

    # Создание файла правил, если его нет
    if not os.path.exists(args.rules):
        with open(args.rules, 'w', encoding='utf-8') as f:
            f.write(SAMPLE_RULES)
        print(f"[*] Створений файл правил: {args.rules}")
```

```
    if args.api and not FLASK_AVAILABLE:
        print("[!] Flask не встановлений")
        args.api = False

    notifier = TelegramNotifier()

    ids_instance = IDS(interface=args.interface,
rules_path=args.rules, enable_ips=args.ips, api=args.api,
notifier=notifier)

    try:
        ids_instance.start()
    except KeyboardInterrupt:
        pass
    except Exception as e:
        print(f"\n[!] Критична помилка: {e}")
```